Quantitative Information Flow – Verification Hardness and Possibilities

Hirotoshi Yasuoka
Graduate School of Information Sciences
Tohoku University
Sendai, Japan
yasuoka@kb.ecei.tohoku.ac.jp

Tachio Terauchi
Graduate School of Information Sciences
Tohoku University
Sendai, Japan
terauchi@ecei.tohoku.ac.jp

***Abstract*—Researchers have proposed formal definitions of quantitative information flow based on information theoretic notions such as the Shannon entropy, the min entropy, the guessing entropy, and channel capacity. This paper investigates the hardness and possibilities of precisely checking and inferring quantitative information flow according to such definitions.**

We prove that, even for just comparing two programs on which has the larger flow, none of the definitions is a k-safety property for any k, and therefore is not amenable to the self-composition technique that has been successfully applied to precisely checking non-interference. We also show a complexity theoretic gap with non-interference by proving that, for loop-free boolean programs whose non-interference is coNP-complete, the comparison problem is #P-hard for all of the definitions.

For positive results, we show that universally quantifying the distribution in the comparison problem, that is, comparing two programs according to the entropy based definitions on which has the larger flow for all distributions, is a 2-safety problem in general and is coNP-complete when restricted for loop-free boolean programs. We prove this by showing that the problem is equivalent to a simple relation naturally expressing the fact that one program is more secure than the other. We prove that the relation also refines the channel-capacity based definition, and that it can be precisely checked via the self-composition as well as the "interleaved" self-composition technique.

I. Introduction

We consider programs containing high security inputs and low security outputs. Informally, the quantitative information flow problem concerns the amount of information that an attacker can learn about the high security input by executing the program and observing the low security output. The problem is motivated by applications in information security. We refer to the classic by Denning [12] for an overview.

In essence, quantitative information flow measures *how* secure, or insecure, a program is. Thus, unlike non-interference [14], that only tells whether a program is completely secure or not completely secure, a definition of quantitative information flow must be able to distinguish two programs that are both interferent but have different degrees of "secureness."

For example, consider the following two programs:

$$M_1 \equiv \texttt{if}\ H = g\ \texttt{then}\ O\ \texttt{:=}\ 0\ \texttt{else}\ O\ \texttt{:=}\ 1$$
$$M_2 \equiv O\ \texttt{:=}\ H$$

In both programs, H is a high security input and O is a low security output. Viewing H as a password, M_1 is a prototypical login program that checks if the guess g matches the password.[1] By executing M_1, an attacker only learns whether H is equal to g, whereas she would be able to learn the entire content of H by executing M_2. Hence, a reasonable definition of quantitative information flow should assign a higher quantity to M_2 than to M_1, whereas non-interference would merely say that M_1 and M_2 are both interferent, assuming that there are more than one possible value of H.

Researchers have attempted to formalize the definition of quantitative information flow by appealing to information theory. This has resulted in definitions based on the Shannon entropy [12], [7], [19], the min entropy [29], the guessing entropy [16], [1], and channel capacity [22], [20], [26]. Much of the previous research has focused on information theoretic properties of the definitions and approximate (i.e., incomplete and/or unsound) algorithms for checking and inferring quantitative information flow according to such definitions.

In this paper, we give a verification theoretic and complexity theoretic analysis of quantitative information flow and investigate precise methods for checking quantitative information flow. In particular, we study the following *comparison problem*: Given two programs M_1 and M_2, decide if $\mathcal{X}(M_1) \leq \mathcal{X}(M_2)$. Here $\mathcal{X}(M)$ denotes the information flow quantity of the program M according to the quantitative information flow definition $\mathcal{X}$ where $\mathcal{X}$ is either $SE[\mu]$ (Shannon-entropy based with distribution μ), $ME[\mu]$ (min-entropy based with distribution μ), $GE[\mu]$ (guessing-entropy based with distribution μ), or CC (channel-capacity based). Note that, obviously, the comparison problem is no harder than actually computing the quantitative information flow as we can compare the two numbers once we have computed $\mathcal{X}(M_1)$ and $\mathcal{X}(M_2)$.

Concretely, we show the following negative results, where $\mathcal{X}$ is CC, $SE[\mu]$, $ME[\mu]$, or $GE[\mu]$ with μ uniform.

- Checking if $\mathcal{X}(M_1) \leq \mathcal{X}(M_2)$ is not a k-safety

[1] Here, for simplicity, we assume that g is a program constant. See Section II for modeling attacker/user (i.e., low security) inputs.

property [30], [9] for any k.
- Restricted to loop-free boolean programs, checking if $\mathcal{X}(M_1) \leq \mathcal{X}(M_2)$ is #P-hard.

The results are in stark contrast to non-interference which is known to be a 2-safety property in general [3], [11] (technically, for the termination-insensitive case[2]) and can be shown to be coNP-complete for loop-free boolean programs (proved in Section III-C). (#P is known to be as hard as the entire polynomial hierarchy [31].) The results suggest that precisely inferring (i.e., computing) quantitative information flow according to these definitions would be harder than checking non-interference and may require a very different approach (i.e., not self composition [3], [11], [30]).

We also give the following positive results which show checking if the quantitative information flow of one program is larger than the other for all distributions according to the entropy-based definitions is easier. Below, $\mathcal{Y}$ is SE, ME, or GE.

- Checking if $\forall \mu. \mathcal{Y}[\mu](M_1) \leq \mathcal{Y}[\mu](M_2)$ is a 2-safety property.
- Restricted to loop-free boolean programs, checking if $\forall \mu. \mathcal{Y}[\mu](M_1) \leq \mathcal{Y}[\mu](M_2)$ is coNP-complete.

These results are proven by showing that the problems $\forall \mu. SE[\mu](M_1) \leq SE[\mu](M_2)$, $\forall \mu. ME[\mu](M_1) \leq ME[\mu](M_2)$, and $\forall \mu. GE[\mu](M_1) \leq GE[\mu](M_2)$ are all actually equivalent to a simple 2-safety relation $R(M_1, M_2)$. We also show that this relation refines the channel-capacity based quantitative information flow, that is, if $R(M_1, M_2)$ then $CC(M_1) \leq CC(M_2)$.

The fact that $R(M_1, M_2)$ is a 2-safety property implies that it can be reduced to a safety problem via self composition. This leads to a new approach to precisely checking quantitative information flow that leverages recent advances in automated software verification [2], [15], [24], [4]. Briefly, given M_1 and M_2, $R(M_1, M_2)$ means that M_1 is at least as secure as M_2 for all distributions while $\neg R(M_1, M_2)$ means that there must be a distribution in which M_1 is less secure than M_2, according to the entropy-based definitions of quantitative information flow. Therefore, by deciding $R(M_1, M_2)$, we can measure the security of the program M_1 relative to another *specification* program M_2. Note that this is useful even when M_1 and M_2 are "incomparable" by R, that is, when $\neg R(M_1, M_2)$ and $\neg R(M_2, M_1)$. See Section IV-B for the details.

The rest of the paper is organized as follows. Section II reviews the existing information-theoretic definitions of quantitative information flow. Section III proves the hardness of their comparison problems and thus shows the hardness of precisely inferring quantitative information flow according to these definitions. Section IV introduces the relation R and proves it equivalent to the comparison problems for the entropy-based definitions with their distributions universally quantified. The section also shows that this is a 2-safety property and is easier to decide than the non-universally-quantified comparison problems, and suggests a self-composition based method for precisely checking quantitative information flow. Section V discusses related work, and Section VI concludes. Appendix A contains the supporting lemmas and definitions for the proofs appearing in the main text. The omitted proofs appear in Appendix B.

[2]We restrict to terminating programs in this paper. (The termination assumption is nonrestrictive because we assume safety verification as a blackbox routine.)

II. Preliminaries

We introduce the information theoretic definitions of quantitative information flow that have been proposed in literature. First, we review the notion of the *Shannon entropy* [28], $\mathcal{H}[\mu](X)$, which is the average of the information content, and intuitively, denotes the uncertainty of the random variable X.

Definition 2.1 (Shannon Entropy): *Let X be a random variable with sample space $\mathbb{X}$ and μ be a probability distribution associated with X (we write μ explicitly for clarity). The Shannon entropy of X is defined as*

$$\mathcal{H}[\mu](X) = \sum_{x \in \mathbb{X}} \mu(X = x) \log \frac{1}{\mu(X = x)}$$

(The logarithm is in base 2.)

Next, we define *conditional entropy*. Informally, the conditional entropy of X given Y denotes the uncertainty of X after knowing Y.

Definition 2.2 (Conditional Entropy): *Let X and Y be random variables with sample spaces $\mathbb{X}$ and $\mathbb{Y}$, respectively, and μ be a probability distribution associated with X and Y. Then, the conditional entropy of X given Y, written $\mathcal{H}[\mu](X|Y)$ is defined as*

$$\mathcal{H}[\mu](X|Y) = \sum_{y \in \mathbb{Y}} \mu(Y = y) \mathcal{H}[\mu](X|Y = y)$$

where

$$\begin{aligned} &\mathcal{H}[\mu](X|Y = y) \\ &\quad = \textstyle\sum_{x \in \mathbb{X}} \mu(X = x|Y = y) \log \frac{1}{\mu(X = x|Y = y)} \\ &\mu(X = x|Y = y) = \tfrac{\mu(X = x, Y = y)}{\mu(Y = y)} \end{aligned}$$

Next, we define (conditional) mutual information. Intuitively, the conditional mutual information of X and Y given Z represents the mutual dependence of X and Y after knowing Z.

Definition 2.3 (Mutual Information): *Let X, Y and Z be random variables and μ be an associated probability distribution.*[3] *Then, the conditional mutual information of X and Y given Z is defined as*

$$\begin{aligned} \mathcal{I}[\mu](X; Y|Z) &= \mathcal{H}[\mu](X|Z) - \mathcal{H}[\mu](X|Y, Z) \\ &= \mathcal{H}[\mu](Y|Z) - \mathcal{H}[\mu](Y|X, Z) \end{aligned}$$

[3]We abbreviate sample spaces of random variables when they are clear from the context.

Let M be a program that takes a high security input H and a low security input L, and gives the low security output O. For simplicity, we restrict to programs with just one variable of each kind, but it is trivial to extend the formalism to multiple variables (e.g., by letting the variables range over tuples). Also, for the purpose of the paper, unobservable (i.e., high security) outputs are irrelevant, and so we assume that the only program output is the low security output. Let μ be a probability distribution over the values of H and L. Then, the semantics of M can be defined by the following probability equation. (We restrict to terminating deterministic programs in this paper.)

$$\mu(O = o) = \sum_{\substack{h,\, \ell \,\in\, \mathbb{H},\, \mathbb{L} \\ M(h,\ell) \,=\, o}} \mu(H = h, L = \ell)$$

Note that we write $M(h, \ell)$ to denote the low security output of the program M given inputs h and ℓ. Now, we are ready to introduce the Shannon-entropy based definition of quantitative information flow (QIF) [12], [7], [19].

Definition 2.4 (Shannon-Entropy-based QIF): *Let M be a program with high security input H, low security input L, and low security output O. Let μ be a distribution over H and L. Then, the Shannon-entropy-based quantitative information flow is defined*

$$\begin{array}{rcl} SE[\mu](M) & = & \mathcal{I}[\mu](O; H|L) \\ & = & \mathcal{H}[\mu](H|L) - \mathcal{H}[\mu](H|O, L) \end{array}$$

Intuitively, $\mathcal{H}[\mu](H|L)$ denotes the initial uncertainty knowing the low security input and $\mathcal{H}[\mu](H|O, L)$ denotes the remaining uncertainty after knowing the low security output.

As an example, consider the programs M_1 and M_2 from Section I. For concreteness, assume that g is the value 01 and H ranges over the space $\{00, 01, 10, 11\}$. Let U be the uniform distribution over $\{00, 01, 10, 11\}$, that is, $U(h) = 1/4$ for all $h \in \{00, 01, 10, 11\}$. The results are as follows.

$$\begin{array}{rl} SE[U](M_1) & = \mathcal{H}[U](H) - \mathcal{H}[U](H|O) \\ & = \log 4 - \frac{3}{4} \log 3 \\ & \approx .81128 \end{array}$$

$$\begin{array}{rl} SE[U](M_2) & = \mathcal{H}[U](H) - \mathcal{H}[U](H|O) \\ & = \log 4 - \log 1 \\ & = 2 \end{array}$$

Consequently, we have that $SE[U](M_1) \leq SE[U](M_2)$, but $SE[U](M_2) \not\leq SE[U](M_1)$. That is, M_1 is more secure than M_2 (according to the Shannon-entropy based definition with uniformly distributed inputs), which agrees with our intuition.

Let us recall the notion of non-interference [10], [14].

Definition 2.5 (Non-intereference): *A program M is said to be non-interferent iff for any $h, h' \in \mathbb{H}$ and $\ell \in \mathbb{L}$, $M(h, \ell) = M(h', \ell)$.*

It is worth noting that non-interference can be formalized as a special case of the Shannon-entropy based quantitative information flow where the flow quantity is zero.

Theorem 2.6: *Let M be a program that takes high-security input H, low-security input L, and returns low-security output O. Then, M is non-interferent if and only if $\forall \mu. SE[\mu](M) = 0$.*

The above theorem is complementary to the one proven by Clark et al. [5] which states that for any μ such that $\mu(H = h, L = \ell) > 0$ for all $h \in \mathbb{H}$ and $\ell \in \mathbb{L}$, $SE[\mu](M) = 0$ iff M is non-interferent.

Next, we introduce the *min entropy*, which Smith [29] recently suggested as an alternative measure for quantitative information flow.

Definition 2.7 (Min Entropy): *Let X and Y be random variables, and μ be an associated probability distribution. Then, the min entropy of X is defined*

$$\mathcal{H}_\infty[\mu](X) = \log \frac{1}{\mathcal{V}[\mu](X)}$$

and the conditional min entropy of X given Y is defined

$$\mathcal{H}_\infty[\mu](X|Y) = \log \frac{1}{\mathcal{V}[\mu](X|Y)}$$

where

$$\begin{array}{rl} \mathcal{V}[\mu](X) & = \max_{x \in \mathbb{X}} \mu(X = x) \\ \mathcal{V}[\mu](X|Y = y) & = \max_{x \in \mathbb{X}} \mu(X = x | Y = y) \\ \mathcal{V}[\mu](X|Y) & = \sum_{y \in \mathbb{Y}} \mu(Y = y) \mathcal{V}[\mu](X|Y = y) \end{array}$$

Intuitively, $\mathcal{V}[\mu](X)$ represents the highest probability that an attacker guesses X in a single try. We now define the min-entropy-based definition of quantitative information flow.

Definition 2.8 (Min-Entropy-based QIF): *Let M be a program with high security input H, low security input L, and low security output O. Let μ be a distribution over H and L. Then, the min-entropy-based quantitative information flow is defined*

$$ME[\mu](M) = \mathcal{H}_\infty[\mu](H|L) - \mathcal{H}_\infty[\mu](H|O, L)$$

Whereas Smith [29] focused on programs lacking low security inputs, we extend the definition to programs with low security inputs in the definition above. It is easy to see that our definition coincides with Smith's for programs without low security inputs. Also, the extension is arguably natural in the sense that we simply take the conditional entropy with respect to the distribution over the low security inputs.

Computing the min-entropy based quantitative information flow for our running example programs M_1 and M_2

from Section I with the uniform distribution, we obtain,

$$\begin{aligned} ME[U](M_1) &= \mathcal{H}_\infty[U](H) - \mathcal{H}_\infty[U](H|O) \\ &= \log 4 - \log 2 \\ &= 1 \end{aligned}$$

$$\begin{aligned} ME[U](M_2) &= \mathcal{H}_\infty[U](H) - \mathcal{H}_\infty[U](H|O) \\ &= \log 4 - \log 1 \\ &= 2 \end{aligned}$$

Again, we have that $ME[U](M_1) \leq ME[U](M_2)$ and $ME[U](M_2) \not\leq ME[U](M_1)$, and so M_2 is deemed less secure than M_1.

The third definition of quantitative information flow treated in this paper is the one based on the guessing entropy [21], that is also recently proposed in literature [16], [1].

Definition 2.9 (Guessing Entropy): *Let X and Y be random variables, and μ be an associated probability distribution. Then, the guessing entropy of X is defined*

$$\mathcal{G}[\mu](X) = \sum_{1 \leq i \leq m} i \times \mu(X = x_i)$$

where $\{x_1, x_2, \ldots, x_m\} = \mathbb{X}$ and $\forall i, j.i \leq j \Rightarrow \mu(X = x_i) \geq \mu(X = x_j)$.

The conditional guessing entropy of X given Y is defined

$$\mathcal{G}[\mu](X|Y) = \sum_{y \in \mathbb{Y}} \mu(Y = y) \sum_{1 \leq i \leq m} i \times \mu(X = x_i | Y = y)$$

where $\{x_1, x_2, \ldots, x_m\} = \mathbb{X}$ and $\forall i, j.i \leq j \Rightarrow \mu(X = x_i|Y = y) \geq \mu(X = x_j|Y = y)$.

Intuitively, $\mathcal{G}[\mu](X)$ represents the average number of times required for the attacker to guess the value of X. We now define the guessing-entropy-based quantitative information flow.

Definition 2.10 (Guessing-Entropy-based QIF): *Let M be a program with high security input H, low security input L, and low security output O. Let μ be a distribution over H and L. Then, the guessing-entropy-based quantitative information flow is defined*

$$GE[\mu](M) = \mathcal{G}[\mu](H|L) - \mathcal{G}[\mu](H|O, L)$$

Like with the min-entropy-based definition, the previous research on guessing-entropy-based quantitative information flow only considered programs without low security inputs [16], [1]. But, it is easy to see that our definition with low security inputs coincides with the previous definitions for programs without low security inputs. Also, as with the extension for the min-entropy-based definition, it simply takes the conditional entropy over the low security inputs.

We test *GE* on the running example from Section I by calculating the quantities for the programs M_1 and M_2 with the uniform distribution.

$$\begin{aligned} GE[U](M_1) &= \mathcal{G}[U](H) - \mathcal{G}[U](H|O) \\ &= \tfrac{5}{2} - \tfrac{7}{4} \\ &= 0.75 \end{aligned}$$

$$\begin{aligned} GE[U](M_2) &= \mathcal{G}[U](H) - \mathcal{G}[U](H|O) \\ &= \tfrac{5}{2} - 1 \\ &= 1.5 \end{aligned}$$

Therefore, we again have that $GE[U](M_1) \leq GE[U](M_2)$ and $GE[U](M_2) \not\leq GE[U](M_1)$, and so M_2 is considered less secure than M_1, even with the guessing-entropy based definition with the uniform distribution.

The fourth and the final existing definition of quantitative information flow that we introduce in this paper is the one based on *channel capacity* [22], [20], [26], which is simply defined to be the maximum of the Shannon-entropy based quantitative information flow over the distribution.

Definition 2.11 (Channel-Capacity-based QIF): *Let M be a program with high security input H, low security input L, and low security output O. Then, the channel-capacity-based quantitative information flow is defined*

$$CC(M) = \max_{\mu} \mathcal{I}[\mu](O; H|L)$$

Unlike the Shannon-entropy based, the min-entropy based, and the guessing-entropy based definitions, the channel-capacity based definition of quantitative information flow is not parameterized by a distribution over the inputs. As with the other definitions, let us test the definition on the running example from Section I by calculating the quantities for the programs M_1 and M_2:

$$\begin{aligned} CC(M_1) &= \max_\mu \mathcal{I}[\mu](O; H) \\ &= 1 \end{aligned}$$

$$\begin{aligned} CC(M_2) &= \max_\mu \mathcal{I}[\mu](O; H) \\ &= 2 \end{aligned}$$

As with the entropy-based definitions (with the uniform distribution), we have that $CC(M_1) \leq CC(M_2)$ and $CC(M_2) \not\leq CC(M_1)$, that is, the channel-capacity based quantitative information flow also says that M_2 is less secure than M_1.

III. Hardness of Comparison Problems

We investigate the hardness of deciding the following *comparison problem* $C_{SE}[\mu]$: Given programs M_1 and M_2 having the same input domain, decide if $SE[\mu](M_1) \leq SE[\mu](M_2)$. Because we are interested in hardness, we focus on the case where μ is the uniform distribution U. That is, the results we prove for the specific case applies to the general case. Also note that the comparison problem is no harder than actually computing the quantitative information flow because we can compare $SE[\mu](M_1)$ and $SE[\mu](M_2)$ if we know their actual values.

Likewise, we study the hardness of the comparison problem $C_{ME}[\mu]$, defined to be the problem $ME[\mu](M_1) \leq ME[\mu](M_2)$, $C_{GE}[\mu]$, defined to be the problem $GE[\mu](M_1) \leq GE[\mu](M_2)$, and C_{CC}, defined to be the problem $CC(M_1) \leq CC(M_2)$. As with $C_{SE}[\mu]$, we require the two programs to share the same input domain for these problems.

We show that none of these comparison problems are k-safety problems for any k. Informally, a program property is said to be a *k-safety* property [30], [9] if it can be refuted by observing k number of (finite) execution traces. A k-safety problem is the problem of checking a k-safety property. Note that the standard safety property is a 1-safety property. An important property of a k-safety problem is that it can be reduced to a standard safety (i.e., 1-safety) problem, such as the unreachability problem, via a simple program transformation called *self composition* [3], [11].

It is well-known that non-interference is a 2-safety property,[4] and this has enabled its precise checking via a reduction to a safety problem via self composition and piggybacking on advances in automated safety verification methods [30], [25], [32]. Unfortunately, the results in this section imply that quantitative information flow inference problem is unlikely to receive the same benefits.

Because we are concerned with properties about pairs of programs (i.e., comparison problems), we extend the notion of k-safety to properties refutable by observing k traces from each of the two programs. More formally, we say that the comparison problem C is a k-safety property if $(M_1, M_2) \notin C$ implies that there exists $T_1 \subseteq [\![M_1]\!]$ and $T_2 \subseteq [\![M_2]\!]$ such that

(1) $|T_1| \leq k$
(2) $|T_2| \leq k$
(3) $\forall M_1', M_2'. T_1 \subseteq [\![M_1']\!] \wedge T_2 \subseteq [\![M_2']\!] \Rightarrow (M_1', M_2') \notin C$

In the above, $[\![M]\!]$ denotes the semantics (i.e., traces) of M, represented by the set of input/output pairs $\{((h, \ell), o) \mid h \in \mathbb{H}, \ell \in \mathbb{L}, o = M(h, \ell)\}$.

We now state the main results of the section. (Recall that U denotes the uniform distribution.) We sketch the main idea of the proofs. All proofs are by contradiction. Let C be the comparison problem in the statement and suppose C is k-safety. Let $(M_1, M_2) \notin C$. Then, we have $T_1 \subseteq [\![M_1]\!]$ and $T_2 \subseteq [\![M_2]\!]$ satisfying the properties (1), (2), and (3) above. From this, we construct $\bar{M}_1$ and $\bar{M}_2$ such that $T_1 \subseteq [\![\bar{M}_1]\!]$ and $T_2 \subseteq [\![\bar{M}_2]\!]$ and $(\bar{M}_1, \bar{M}_2) \in C$ to obtain the contradiction.

Theorem 3.1: *$C_{SE}[U]$ is not a k-safety property for any $k > 0$.*

Theorem 3.2: *$C_{ME}[U]$ is not a k-safety property for any $k > 0$.*

Theorem 3.3: *$C_{GE}[U]$ is not a k-safety property for any $k > 0$.*

[4]It is also well known that it is not a 1-safety property [23].

Theorem 3.4: *C_{CC} is not a k-safety property for any $k > 0$.*

A. Bounding the Domains

The notion of k-safety property, like the notion of safety property from where it extends, is defined over all programs regardless of their size. (For example, non-interference is a 2-safety property for all programs and unreachability is a safety property for all programs.) But, it is easy to show that the comparison problems would become "k-safety" properties if we constrained and bounded the input domains because then the size of the semantics (i.e., the input/output pairs) of such programs would be bounded by $|\mathbb{H}| \times |\mathbb{L}|$. In this case, the problems are at most $|\mathbb{H}| \times |\mathbb{L}|$-safety.[5] However, these bounds are high for all but very small domains, and are unlikely to lead to a practical verification method.

B. Proof of Theorem 3.1

We discuss the details of the proof of Theorem 3.1. The proofs of Theorems 3.2, 3.3, 3.4 are deferred to Appendix B.

For contradiction, suppose $C_{SE}[U]$ is a k-safety property. Let M and M' be programs having the same input domain such that $(M, M') \notin C_{SE}[U]$. Then, it must be the case that there exist $T \subseteq [\![M]\!]$ and $T' \subseteq [\![M']\!]$ such that $|T| \leq k$, $|T'| \leq k$, and $\forall M_c, M_c'. T \subseteq [\![M_c]\!] \wedge T' \subseteq [\![M_c']\!] \Rightarrow (M_c, M_c') \notin C_{SE}[U]$.

Let

$$T = \{(h_1, o_1), (h_2, o_2), \ldots, (h_i, o_i)\}$$
$$T' = \{(h_1', o_1'), (h_2', o_2'), \ldots, (h_j', o_j')\}$$

where $i, j \leq k$. Now, we construct new programs $\bar{M}$ and $\bar{M}'$ as follows.

$$\begin{array}{ll}
\bar{M}(h_1) = o_1 & \bar{M}'(h_1') = o_1' \\
\bar{M}(h_2) = o_2 & \bar{M}'(h_2') = o_2' \\
\cdots & \cdots \\
\bar{M}(h_i) = o_i & \bar{M}'(h_j') = o_j' \\
\bar{M}(h_{i+1}) = o & \bar{M}'(h_{j+1}') = o_{j+1}' \\
\bar{M}(h_{i+2}) = o & \bar{M}'(h_{j+2}') = o_{j+2}' \\
\cdots & \cdots \\
\bar{M}(h_{i+j}) = o & \bar{M}'(h_{j+i}') = o_{j+i}' \\
\bar{M}(h_{i+j+1}) = o_r & \bar{M}'(h_{j+i+1}') = o_r' \\
\cdots & \cdots \\
\bar{M}(h_n) = o_r & \bar{M}'(h_n') = o_r'
\end{array}$$

where

- $o \neq o_r$,
- $\{o_1, o_2, \ldots, o_i\} \cap \{o, o_r\} = \emptyset$,
- o_{j+1}', o_{j+2}', …, o_{j+i}', and o_r' are distinct,
- $\{o_1', o_2', \ldots, o_j'\} \cap \{o_{j+1}', \ldots, o_{j+i}', o_r'\} = \emptyset$,
- $\{h_1, \ldots, h_n\} = \{h_1', \ldots, h_n'\}$, and
- $n = 2k$.

[5]It is possible to get a tighter bound for the channel-capacity based definition by also bounding the size of the output domain.

$$
\begin{array}{rcl}
M & ::= & x := \psi \mid \mathsf{if}\ \psi\ \mathsf{then}\ M\ \mathsf{else}\ M \mid M_0; M_1 \\
\phi, \psi & ::= & \mathsf{true} \mid x \mid \phi \wedge \psi \mid \neg\phi
\end{array}
$$

Figure 1. The syntax of loop-free boolean programs

$$
\begin{array}{l}
\mathit{wp}(x := \psi, \phi) = \phi[\psi/x] \\
\mathit{wp}(\mathsf{if}\ \psi\ \mathsf{then}\ M_0\ \mathsf{else}\ M_1, \phi) \\
\quad = (\psi \Rightarrow \mathit{wp}(M_0, \phi)) \wedge (\neg\psi \Rightarrow \mathit{wp}(M_1, \phi)) \\
\mathit{wp}(M_0; M_1, \phi) = \mathit{wp}(M_0, \mathit{wp}(M_1, \phi))
\end{array}
$$

Figure 2. The weakest precondition for loop-free boolean programs

Then, comparing the Shannon-entropy-based quantitative information flow of $\bar{M}$ and $\bar{M}'$, we have,

$$
\begin{array}{l}
SE[U](\bar{M}') - SE[U](\bar{M}) \\
\quad = \sum_{o'_x \in \{o'_1, \ldots, o'_i\}} U(o'_x) \log \frac{1}{U(o'_x)} \\
\qquad + U(o') \log \frac{1}{U(o')} + U(o'_r) \log \frac{1}{U(o'_r)} \\
\qquad - (\sum_{o_x \in \{o_1, \ldots, o_j\}} U(o_x) \log \frac{1}{U(o_x)} \\
\qquad\quad + \sum_{o_y \in \{o_{j+1}, \ldots, o_{j+i}\}} U(o_y) \log \frac{1}{U(o_y)} \\
\qquad\qquad + U(o_r) \log \frac{1}{U(o_r)})
\end{array}
$$

(Note the abbreviations from Appendix A.) By lemma A.5, we have

$$
\begin{array}{l}
\sum_{o_x \in \{o_1, \ldots, o_i\}} U(o_x) \log \frac{1}{U(o_x)} \\
\qquad \leq \sum_{o'_y \in \{o'_{j+1}, \ldots, o'_{j+i}\}} U(o'_y) \log \frac{1}{U(o'_y)}
\end{array}
$$

and

$$
U(o) \log \frac{1}{U(o)} \leq \sum_{o'_x \in \{o'_1, \ldots, o'_j\}} U(o'_x) \log \frac{1}{U(o'_x)}
$$

Trivially, we have

$$
U(o'_r) \log \frac{1}{U(o'_r)} = U(o_r) \log \frac{1}{U(o_r)}
$$

As a result, we have

$$
SE[U](\bar{M}') - SE[U](\bar{M}) \geq 0
$$

Note that $\bar{M}$ and $\bar{M}'$ have the same counterexamples T and T', that is, $T \subseteq [\![\bar{M}]\!]$ and $T' \subseteq [\![\bar{M}']\!]$. However, we have $(\bar{M}, \bar{M}') \in C_{SE}[U]$. This leads to a contradiction.

C. Complexities for Loop-free Boolean Programs

The purpose of this section is to show a complexity theoretic gap between non-interference and quantitative information flow. The results strengthen the hypothesis that quantitative information flow is quite hard to compute precisely, and also suggest an interesting connection to counting problems.

We focus on loop-free boolean programs whose syntax is given in Figure 1. We assume the usual derived formulas $\phi \Rightarrow \psi$, $\phi = \psi$, $\phi \vee \psi$, and false. We give the usual weakest precondition semantics in Figure 2.

To adapt the information flow framework to boolean programs, we make each information flow variable H, L, and O range over functions mapping boolean variables of its kind to boolean values. So, for example, if x and y are low security boolean variables and z is a high security boolean variable, then L ranges over the functions $\{x, y\} \to \{\mathsf{false}, \mathsf{true}\}$, and H and O range over $\{z\} \to \{\mathsf{false}, \mathsf{true}\}$.[6] (Every boolean variable is either a low security boolean variable or a high security boolean variable.) We write $M(h, \ell) = o$ for an input (h, ℓ) and an output o if $(h, \ell) \models \mathit{wp}(M, \phi)$ for a boolean formula ϕ such that $o \models \phi$ and $o' \not\models \phi$ for all output $o' \neq o$. Here, $\models$ is the usual logical satisfaction relation, using h, ℓ, o, etc. to look up the values of the boolean variables. (Note that this incurs two levels of lookup.)

As an example, consider the following program.

$$
\begin{array}{l}
M \equiv \\
\quad z := x; w := y; \\
\quad \mathsf{if}\ x \wedge y\ \mathsf{then}\ z := \neg z\ \mathsf{else}\ w := \neg w
\end{array}
$$

Let x, y and w be high security variables and z be a low security variable. Then,

$$
\begin{array}{rcl}
SE[U](M) & = & 1.5 \\
ME[U](M) & = & \log 3 \\
 & \approx & 1.5849625 \\
GE[U](M) & = & 1.25 \\
CC(M) & = & \log 3 \\
 & \approx & 1.5849625
\end{array}
$$

We prove the following hardness results. These results are proven by a reduction from #SAT, which is the problem of counting the number of solutions to a quantifier-free boolean formula. #SAT is known to be #P-complete. Because #SAT is a function problem and the comparison problems are decision problems, a step in the proofs makes binary search queries to the comparison problem oracle a polynomial number of times. (Recall that the notation FP^A means the complexity class of function problems solvable in polynomial time with an oracle for the problem A.)

Theorem 3.5: $\#P \subseteq FP^{C_{SE}[U]}$

Theorem 3.6: $\#P \subseteq FP^{C_{ME}[U]}$

Theorem 3.7: $\#P \subseteq FP^{C_{GE}[U]}$

Theorem 3.8: $\#P \subseteq FP^{C_{CC}}$

We remind that the above results apply (even) when the comparison problems $C_{SE}[U]$, $C_{ME}[U]$, $C_{GE}[U]$, and C_{CC} are restricted to loop-free boolean programs.

In summary, each comparison problem $C_{SE}[U]$, $C_{ME}[U]$, $C_{GE}[U]$, and C_{CC} can be used a polynomial number of times to solve a #P-complete problem. Because Toda's theorem [31] implies that the entire polynomial hierarchy can be solved by using a #P-complete oracle a polynomial number of times, our results show that the comparison problems for quantitative information flow can also be used

[6] We do not distinguish input boolean variables from output boolean variables. But, a boolean variable can be made output-only by assigning a constant to the variable at the start of the program and made input-only by assigning a constant at the end.

a polynomial number of times to solve the entire polynomial hierarchy, for the case of loop-free boolean programs.

As shown below, this presents a gap from non-interference, which is only coNP-complete for loop-free boolean programs.

Theorem 3.9: *Checking non-interference is coNP-complete for loop-free boolean programs.*

The above is an instance of the general observation that, by solving quantitative information flow problems, one is able to solve the class of problems known as *counting problems*,[7] which coincides with #SAT for the case of loop-free boolean programs.

D. Proof of Theorem 3.5

We discuss the details of the proof of Theorem 3.5. The proofs of Theorems 3.6, 3.7, 3.8 are deferred to Appendix B.

First, we prove the following lemma which states that we can compare the number of solutions to boolean formulas by computing $SE[U]$. (For convenience, we use large letters H, L, O, etc. to range over boolean variables as well as generic random variables.)

Lemma 3.10: *Let $\overrightarrow{H}$ and H' be distinct boolean random variables. Let i and j be any non-negative integers such that $i \leq 2^{|\overrightarrow{H}|}$ and $j \leq 2^{|\overrightarrow{H}|}$. Let ψ_i (resp. ψ_j) be a formula over $\overrightarrow{H}$ having i (resp. j) assignments. Then, $j \leq i$ iff $SE[U](M_j) \leq SE[U](M_i)$ where $M_j \equiv O := \psi_j \wedge H'$ and $M_i \equiv O := \psi_i \wedge H'$.*

Proof: Let $p = \frac{i}{2^{|H|+1}}$ and $q = \frac{j}{2^{|H|+1}}$. We have

$$\begin{aligned} SE[U](M_j) &= \tfrac{j}{2^{|H|+1}} \log \tfrac{2^{|H|+1}}{j} + \tfrac{2^{|H|+1}-j}{2^{|H|+1}} \log \tfrac{2^{|H|+1}}{2^{|H|+1}-j} \\ &= p \log p + (1-p) \log \tfrac{1}{1-p} \\ SE[U](M_i) &= \tfrac{i}{2^{|H|+1}} \log \tfrac{2^{|H|+1}}{i} + \tfrac{2^{|H|+1}-i}{2^{|H|+1}} \log \tfrac{2^{|H|+1}}{2^{|H|+1}-i} \\ &= q \log q + (1-q) \log \tfrac{1}{1-q} \end{aligned}$$

- **Only If**
  Suppose $j \leq i$. Then,

$$\begin{aligned} &SE[U](M_i) - SE[U](M_j) \\ &\quad = p \log \tfrac{1}{p} + (1-p) \log \tfrac{1}{1-p} \\ &\qquad -q \log \tfrac{1}{q} - (1-q) \log \tfrac{1}{1-q} \\ &\quad = \log (\tfrac{1-p}{p})^p \tfrac{1-q}{1-p} (\tfrac{q}{1-q})^q \end{aligned}$$

  Then, from $\frac{1-q}{1-p} \geq 1$ and $p \geq q \geq 0$, we have

$$\begin{aligned} SE[U](M_i) - SE[U](M_j) &\geq \log (\tfrac{1-p}{p})^p (\tfrac{q}{1-q})^q \\ &\geq \log (\tfrac{1-p}{p})^q (\tfrac{q}{1-q})^q \\ &= \log (\tfrac{(1-p)q}{p(1-q)})^q \\ &= \log (\tfrac{q-pq}{p-pq})^q \\ &= \log (\tfrac{pq-q}{pq-p})^q \\ &= \log (\tfrac{1-\frac{1}{p}}{1-\frac{1}{q}})^q \\ &\geq 0 \end{aligned}$$

  The last line follows from $\frac{1-\frac{1}{p}}{1-\frac{1}{q}} \geq 1$.

- **If**
  We prove the contraposition. Suppose $j > i$. Then,

$$\begin{aligned} &SE[U](M_j) - SE[U](M_i) \\ &\quad = q \log \tfrac{1}{q} + (1-q) \log \tfrac{1}{1-q} \\ &\qquad -p \log \tfrac{1}{p} - (1-p) \log \tfrac{1}{1-p} \\ &\quad > 0 \end{aligned}$$

  The last line follows from the fact that $0 \leq p < q \leq \frac{1}{2}$. Therefore, $SE[U](M_j) \not\leq SE[U](M_i)$.

■

Then, using Lemma 3.10, we prove the following lemma which is crucial to proving Theorem 3.5.

Lemma 3.11: *Let $\overrightarrow{H}$ be distinct variables and ϕ be a boolean formula over $\overrightarrow{H}$. Then, the number of assignments for ϕ can be computed by executing an oracle that decides whether programs are in $C_{SE}[U]$ at most $3 * (|\overrightarrow{H}| + 1) + 2$ times.*

Proof: First, we define a procedure that returns the number of solutions of ϕ.

Let $F(j) \equiv O := \psi \wedge H'$ where ψ is a formula over $\overrightarrow{H}$ having j assignments and H' be a boolean variable such that $H' \notin \{\overrightarrow{H}\}$. Note that, by Lemma A.4, such ψ can be generated in linear time.

Then, we invoke the following procedure where $M' \equiv O' := \phi \wedge H'$.

```
l = 0;
r = 2^|H|;
n = (l + r)/2;
while ¬C_SE[U](F(n), M') ∨ ¬C_SE[U](M', F(n))
    if C_SE[U](F(n), M')
        then {l = n; n = (l + r)/2;}
        else {r = n; n = (l + r)/2;}
return n
```

Note that when the procedure terminates, we have $SE[U](F(n)) = SE[U](M')$, and so by Lemma 3.10, n is the number of satisfying assignments to ϕ.

We show that the procedure iterates at most $|\overrightarrow{H}|+1$ times. To see this, every iteration in the procedure narrows the range between r and ℓ by one half. Because $r - \ell$ is bounded by $2^{|\overrightarrow{H}|}$, it follows that the procedure iterates at most $|\overrightarrow{H}|+1$ times. Hence, the oracle $C_{SE}[U]$ is accessed $3*(|\overrightarrow{H}|+1)+2$ times, and this proves the lemma. ■

Finally, Theorem 3.5 follows from Lemma 3.11 and the fact that #SAT, the problem of counting the number of solutions to a boolean formula, is #P-complete.

IV. Universally Quantifying Distributions

As proved in Section III, precisely computing quantitative information flow is quite difficult. Indeed, we have shown that even just comparing two programs on which has the larger flow is difficult (i.e., C_{SE}, C_{ME}, C_{GE}, and C_{CC}).

[7] Formally, a counting problem is the problem of counting the number of solutions to a decision problem. For instance, #P is the class of counting problems associated with NP.

In this section, we show that universally quantifying the Shannon-entropy based comparison problem $C_{SE}[\mu]$, the min-entropy based problem $C_{ME}[\mu]$, or the guessing-entropy based problem $C_{GE}[\mu]$ over the distribution μ is equivalent to a simple relation R enjoying the following properties.

(1) R is a 2-safety property.

(2) R is coNP-complete for loop-free boolean programs.

Note that (1) implies that we can actually check if $(M_1, M_2) \in C_{SE}[\mu]$ for all μ via self composition (and likewise for $C_{ME}[\mu]$ and $C_{GE}[\mu]$). We actually show in Section IV-B that we can even use the security-type-based approach suggested by Terauchi and Aiken [30] to minimize code duplication during self composition (i.e., do *interleaved* self composition).

We remind that except for the coNP-completeness result (Theorem 4.8), the results in this section apply to any (deterministic and terminating) programs and not just to loop-free boolean programs.

Definition 4.1: *We define R to be the relation such that $R(M_1, M_2)$ iff for all $\ell \in \mathbb{L}$ and $h, h' \in \mathbb{H}$, if $M_1(h, \ell) \neq M_1(h', \ell)$ then $M_2(h, \ell) \neq M_2(h', \ell)$.*

Note that $R(M_1, M_2)$ essentially says that if an attacker can distinguish a pair of high security inputs by executing M_1, then she could do the same by executing M_2. Hence, R naturally expresses that M_1 is at least as secure as M_2. [8]

It may be somewhat surprising that this simple relation is actually equivalent to the rather complex entropy-based quantitative information flow definitions when they are cast as comparison problems and the distributions are universally quantified, as stated in the following theorems. First, we show that R coincides exactly with C_{SE} with its distribution universally quantified.

Theorem 4.2: $R = \{(M_1, M_2) \mid \forall \mu. C_{SE}[\mu](M_1, M_2)\}$

The proof is detailed in Section IV-A. The next two theorems show that R also coincides with C_{ME} and C_{GE} with their distribution universally quantified.

Theorem 4.3: $R = \{(M_1, M_2) \mid \forall \mu. C_{ME}[\mu](M_1, M_2)\}$

Theorem 4.4: $R = \{(M_1, M_2) \mid \forall \mu. C_{GE}[\mu](M_1, M_2)\}$

The first half of the $\subseteq$ direction of the proofs for the theorems above is much like the that of Theorem 4.2, that is, it makes the observation that M_2 disambiguates the high security inputs at least as fine as does M_1. Then, the proof concludes by utilizing the particular mathematical properties relevant to the respective definitions. The proof for the $\supseteq$ direction is also similar to the argument used in Theorem 4.2. The details of the proofs appear in Appendix B.

Next, we show that R refines C_{CC} in the sense that if $R(M_1, M_2)$ then $C_{CC}(M_1, M_2)$.

[8]We note that notions similar to R have appeared in literature (often in somewhat different representations) [27], [18], [6]. In particular, Clark et al. [6] have shown a result analogous to the $\subseteq$ direction of Theorem 4.2 below. But, R's properties have not been fully investigated.

Theorem 4.5: $R \subseteq C_{CC}$

Note that, the other direction, $R \supseteq C_{CC}$, does not hold as R is not always a total order, whereas C_{CC} is. We also show that R is compatible with the notion of non-interference.

Theorem 4.6: *Let M_2 be a non-interferent program. Then, $R(M_1, M_2)$ iff M_1 is also non-interferent and M_1 has the same input domain as M_2.*

Next, we show that R is easier to decide than the non-universally-quantified versions of the comparison problems. First, it is trivial to see from Definition 4.1 that R is a 2-safety property.

Theorem 4.7: *R is a 2-safety property.*

It can be shown that, restricted to loop-free boolean programs, R is coNP-complete. This follows directly from the observation that we can decide R by self composition thanks to its 2-safety property and the fact that, for loop-free boolean programs, self composition reduces the problem to an UNSAT instance.[9]

Theorem 4.8: *Restricted to loop-free boolean programs, R is coNP-complete.*

A. Proof of Theorem 4.2

We discuss the details of the proof of Theorem 4.2. The proofs of Theorems 4.3, 4.4, 4.5 are deferred to Appendix B.

First, we prove the following lemma which says that, if $R(M, M')$ then $SE[U](M')$ is at least as large as $SE[U](M)$ per each low security input $\ell \in \mathbb{L}$.

Lemma 4.9: *Suppose $R(M, M')$, that is, for all h_1, h_2 in $\mathbb{H}$ and ℓ in $\mathbb{L}$, $M'(h_1, \ell) = M'(h_2, \ell) \Rightarrow M(h_1, \ell) = M(h_2, \ell)$. Let $\mathbb{O}$ be the set of the outputs of M, and $\mathbb{O}'$ be the set of the outputs of M'. Then, for any ℓ, we have $\sum_{o \in \mathbb{O}} \mu(o, \ell) \log \frac{\mu(\ell)}{\mu(o,\ell)} \leq \sum_{o' \in \mathbb{O}'} \mu(o', \ell) \log \frac{\mu(\ell)}{\mu(o',\ell)}$. (Recall the notational convention from Definition A.1.)*

Proof: First, we prove for any output o of M, there exist corresponding outputs $\mathbb{O}_o = \{o'_0, o'_1, \ldots, o'_n\}$ of M' such that

$$\mu(o, \ell) \log \tfrac{\mu(\ell)}{\mu(o,\ell)} \leq \textstyle\sum_{o'_r \in \mathbb{O}_o} \mu(o'_r, \ell) \log \tfrac{\mu(\ell)}{\mu(o'_r,\ell)}$$

Let $\mathbb{H}_o$ be the set such that $\mathbb{H}_o = \{h \mid M(h, \ell) = o\}$. Let $\{h_0, h_1, \ldots, h_n\} = \mathbb{H}_o$. Let $o'_0 = M'(h_0, \ell)$,... and, $o'_n = M'(h_n, \ell)$. For any h' such that $o'_r = M'(h', \ell)$ and $o'_r \in \{o'_0, o'_1, \ldots, o'_n\}$, we have $h' \in \{h_1, \ldots, h_n\}$ since $R(M, M')$. Then, we have $\mu(o, \ell) = \sum_{o'_r \in \{o'_1, \ldots, o'_n\}} \mu(o'_r, \ell)$. By Lemma A.5, we have

$$\mu(o, \ell) \log \tfrac{\mu(\ell)}{\mu(o,\ell)} \leq \textstyle\sum_{o'_r \in \{o'_0, o'_1, \ldots, o'_n\}} \mu(o'_r, \ell) \log \tfrac{\mu(\ell)}{\mu(o'_r,\ell)}$$

[9]To construct a polynomial size boolean formula from a loop-free boolean program, we use the well-known efficient weakest precondition construction technique [13], [17] instead of the naive rules given in Figure 2.

Now to prove the lemma, it suffices to show that each $\mathbb{O}_o$ constructed above are disjoint. That is, for o_1 and o_2 outputs of M such that $o_1 \neq o_2$, $\mathbb{O}_{o_1} \cap \mathbb{O}_{o_2} = \emptyset$. For contradiction, suppose $o' \in \mathbb{O}_{o_1} \cap \mathbb{O}_{o_2}$. Then, there exist h_1 and h_2 such that $o_1 = M(h_1, \ell)$, $o' = M'(h_1, \ell)$, $o_2 = M(h_2, \ell)$, and $o' = M'(h_2, \ell)$. Since $R(M, M')$, we have $o_1 = o_2$, and it leads to a contradiction. Hence, we have

$$\sum_o \mu(o, \ell) \log \frac{\mu(\ell)}{\mu(o, \ell)} \leq \sum_{o'} \mu(o', \ell) \log \frac{\mu(\ell)}{\mu(o', \ell)}$$

■

We now prove Theorem 4.2.

Proof:

- $\subseteq$

  Suppose $(M, M') \in R$. By Lemma A.3,

  $$\begin{aligned} SE[\mu](M) &= \mathcal{H}[\mu](O|L) \\ &= \textstyle\sum_\ell \sum_o \mu(o, \ell) \log \frac{\mu(\ell)}{\mu(o,\ell)} \end{aligned}$$

  and

  $$\begin{aligned} SE[\mu](M') &= \mathcal{H}[\mu](O'|L) \\ &= \textstyle\sum_\ell \sum_{o'} \mu(o', \ell) \log \frac{\mu(\ell)}{\mu(o',\ell)} \end{aligned}$$

  By Lemma 4.9 and the fact that $(M, M') \in R$, we obtain for any ℓ

  $$\sum_o \mu(o, \ell) \log \frac{\mu(\ell)}{\mu(o, \ell)} \leq \sum_{o'} \mu(o', \ell) \log \frac{\mu(\ell)}{\mu(o', \ell)}$$

  Hence,

  $$\begin{aligned} &\textstyle\sum_\ell \sum_o \mu(o, \ell) \log \frac{\mu(\ell)}{\mu(o,\ell)} \\ &\quad \leq \textstyle\sum_\ell \sum_{o'} \mu(o', \ell) \log \frac{\mu(\ell)}{\mu(o',\ell)} \end{aligned}$$

- $\supseteq$

  We prove the contraposition. Suppose $(M, M') \notin R$. Then, there exist o', h_0, h_1, ℓ' such that $o' = M'(h_0, \ell') = M'(h_1, \ell')$ and $M(h_0, \ell') \neq M(h_1, \ell')$. Pick a probability function μ such that $\mu(h_0, \ell') = \mu(h_1, \ell') = \frac{1}{2}$.

  Then, we have

  $$\begin{aligned} H[\mu](O'|L) &= \textstyle\sum_\ell \sum_o \mu(o, \ell) \log \frac{\mu(\ell)}{\mu(o,\ell)} \\ &= \mu(o', \ell') \log \frac{\mu(\ell')}{\mu(o',\ell')} \\ &= 1 \log \frac{1}{1} \\ &= 0 \end{aligned}$$

  Let o_0 and o_1 be output variables such that $o_0 = M(h_0, \ell')$, $o_1 = M(h_1, \ell')$, and $o_0 \neq o_1$.

  $$\begin{aligned} \mathcal{H}[\mu](O|L) &= \textstyle\sum_{o \in \{o_0, o_1\}} \mu(o, \ell') \log \frac{\mu(\ell')}{\mu(o,\ell')} \\ &= \tfrac{1}{2} \log \tfrac{1}{\frac{1}{2}} + \tfrac{1}{2} \log \tfrac{1}{\frac{1}{2}} \\ &= 1 \end{aligned}$$

  Therefore, $SE[\mu](M) \not\leq SE[\mu](M')$, that is, $(M, M') \notin \{(M_1, M_2) \mid \forall \mu. (M_1, M_2) \in C_{SE}[\mu]\}$.

■

B. Quantitative Information Flow via Self Composition

Theorems 4.2, 4.3, 4.4, and 4.7 imply that we can check if the entropy-based quantitative information flow of a program (i.e., *SE*, *ME*, and *GE*) is bounded by that of another for all distributions via self composition [3], [11]. This suggests a novel approach to precisely checking quantitative information flow.

That is, given a *target* program M_1, the user would construct a *specification* program M_2 with the same input domain as M_1 having the desired level of security. Then, she would check $R(M_1, M_2)$ via self composition. If so, then M_1 is guaranteed to be at least as secure as M_2 according to the Shannon-entropy based, the min-entropy based, and the guessing-entropy based definition of quantitative information flow for all distributions (and also channel-capacity based definition), and otherwise, there must be a distribution in which M_1 is less secure than M_2 according to the entropy-based definitions.

Note that deciding $R(M_1, M_2)$ is useful even when M_1 and M_2 are R-incomparable, that is, when neither $R(M_1, M_2)$ nor $R(M_2, M_1)$. This is because $\neg R(M_1, M_2)$ implies that M_1 is less secure than M_2 on some distribution.

For example, suppose M_1 is some complex login program with the high security input H and the low security input L. And we would like to verify that M_1 is at least as secure as the prototypical login program M_2 below.

$$M_2 \equiv \texttt{if } H = L \texttt{ then } O \texttt{ := } 0 \texttt{ else } O \texttt{ := } 1$$

Then, using this framework, it suffices to just query if $R(M_1, M_2)$ is true. (Note that the output domains of M_1 and M_2 need not to match.)

We now describe how to actually check $R(M_1, M_2)$ via self composition. From M_1 and M_2, we construct the self-composed program M' shown below.

$$\begin{array}{l} M'(H, H', L) \equiv \\ \quad O_1 := M_1(H, L); O'_1 := M_1(H', L); \ //\texttt{L1} \\ \quad O_2 := M_2(H, L); O'_2 := M_2(H', L); \ //\texttt{L2} \\ \quad \texttt{assert}(O_1 \neq O'_1 \Rightarrow O_2 \neq O'_2) \end{array}$$

Note that $R(M_1, M_2)$ is true iff M' does not cause an assertion failure. The latter can be checked via a software safety verifier such as SLAM and BLAST [2], [15], [24], [4]. As an aside, we note that this kind of construction could be easily generalized to reduce any k-safety problem (cf. Section III) to a safety problem, as shown by Clarkson and Schneider [9].

Note that the line `L1` (resp. `L2`) of the pseudo code above is M_1 (resp. M_2) sequentially composed with a copy of itself, which is from where the name "self composition" comes. Therefore, technically, M' is a composition of two self compositions.

`L1` (and `L2`) are actually exactly the original self composition proposed for non-interference [3], [11]. Terauchi and Aiken [30] noted that only the parts of M_1 (and M_2) that

depend on the high security inputs H and H' need to be duplicated and self composed, with the rest of the program left intact and "interleaved" with the self-composed parts. The resulting program tends to be verified easier than the naive self composition by modern software safety verifiers.

They proposed a set of transformation rules that translates a WHILE program annotated with security types [33] (or dependency analysis results) to an interleaved self-composed program. This was subsequently improved by a number of researchers to support a richer set of language features and transformation patterns [32], [25]. These transformation methods can be used in place of the naive self compositions at `L1` and `L2` in building M'. That is, we apply a security type inference (or a dependency analysis) to M_1 and M_2 to infer program parts that depend on the high security inputs H and H' so as to only duplicate and self compose those parts of M_1 and M_2.

C. Example

We recall the ideal login program below.

$$M_{spec} \equiv \texttt{if } H = L \texttt{ then } O := 0 \texttt{ else } O := 1$$

We check the following four programs using the above as the specification.

$$M_1 \equiv O := H$$

$$M_2 \equiv \texttt{if } H = L \texttt{ then } O := 0 \texttt{ else } O := H\&1$$

$$\begin{array}{l} M_3 \equiv O := 1; i := 0; \\ \quad \texttt{while } i < 32 \{ \\ \quad\quad m := 1 << i; \\ \quad\quad \texttt{if } H\&m \neq L\&m \texttt{ then} \\ \quad\quad\quad O := 0; \texttt{break}; \\ \quad\quad \texttt{else} \\ \quad\quad\quad i\texttt{++}; \\ \quad \} \end{array}$$

$$\begin{array}{l} M_4 \equiv O := 1; i := 0; \\ \quad \texttt{while } i < 64 \{ \\ \quad\quad m := 1 << i; \\ \quad\quad \texttt{if } H\&m \neq L\&m \texttt{ then} \\ \quad\quad\quad O := 0; \texttt{break}; \\ \quad\quad \texttt{else} \\ \quad\quad\quad i\texttt{++}; \\ \quad \} \end{array}$$

Here, H and L are 64-bit values, & is the bit-wise and operator, and << is the left shift operator. M_1 leaks the entire password. M_2 checks the password against the user guess but then leaks the first bit when the check fails. M_3 only checks the first 32 bits of the password. And, M_4 implements password checking correctly via a while loop.

We verify that only M_4 satisfies the specification, that is, $R(M_4, M_{spec})$. To see that $\neg R(M_1, M_{spec})$, note that for any ℓ, h, h' such that $h \neq \ell$, $h' \neq \ell$ and $h \neq h'$, we have that $M_1(h,\ell) \neq M_1(h',\ell)$ but $M_{spec}(h,\ell) = M_{spec}(h',\ell) = 1$. To see that $\neg R(M_2, M_{spec})$, note that for ℓ, h, h' such that $h \neq \ell$, $h' \neq \ell$, $h\&1 = 1$ and $h'\&1 = 0$, we have that $1 = M_2(h,\ell) \neq M_2(h',\ell) = 0$ but $M_{spec}(h,\ell) = M_{spec}(h',\ell) = 1$. To see that $\neg R(M_3, M_{spec})$, let ℓ, h, h' be such that $h|_{32} = \ell|_{32}$, $h'|_{32} \neq \ell|_{32}$, and $h \neq \ell$, then, $1 = M_3(h,\ell) \neq M_3(h',\ell) = 0$ but $M_{spec}(h,\ell) = M_{spec}(h',\ell) = 1$.[10] (Here, $x|_{32}$ denotes $x \bmod 2^{32}$, i.e., the first 32 bits of x.)

The results imply that for M_1, M_2, and M_3, there must be a distribution where the program is less secure than M_{spec} according to each of the entropy-based definition of quantitative information flow. For instance, for the Shannon-entropy based definition, we have for the uniform distribution U,

$$\begin{array}{rl} SE[U](M_{spec}) & = \frac{1}{2^{58}} + \frac{2^{64}-1}{2^{64}} \log \frac{2^{64}}{2^{64}-1} \\ & \approx 3.46944695 \times 10^{-18} \\ SE[U](M_1) & = 64 \\ SE[U](M_2) & = \frac{1}{2} + \frac{1+2^{63}}{2^{65}} \log \frac{2^{64}}{1+2^{63}} + \frac{2^{63}-1}{2^{65}} \log \frac{2^{64}}{2^{63}-1} \\ & \approx 1.0 \\ SE[U](M_3) & = \frac{1}{2^{27}} + \frac{2^{64}-2^{32}}{2^{64}} \log \frac{2^{64}}{2^{64}-2^{32}} \\ & \approx 7.78648 \times 10^{-9} \end{array}$$

That is, $SE[U](M_1) \not\leq SE[U](M_{spec})$, $SE[U](M_2) \not\leq SE[U](M_{spec})$, and $SE[U](M_3) \not\leq SE[U](M_{spec})$.

Finally, we have that $R(M_4, M_{spec})$, and so M_4 is at least as secure as M_{spec} according to all of the definitions of quantitative information flow considered in this paper. In fact, it can be also shown that $R(M_{spec}, M_4)$. (However, note that M_4 and M_{spec} are not semantically equivalent, i.e., their outputs are reversed.)

V. Related Work

This work builds on previous work that proposed information theoretic notions of quantitative information flow [12], [7], [19], [29], [16], [1], [22], [20], [26]. The previous research has mostly focused on information theoretic properties of the definitions and proposed approximate (i.e., incomplete and/or unsound) methods for checking and inferring them. In contrast, this paper investigates the verification theoretic and complexity theoretic hardness of precisely inferring quantitative information flow according to the definitions and also proposes a precise method for checking quantitative information flow. Our method checks the quantitative information flow of a program against that of a specification program having the desired level of security via self composition for all distributions according to the entropy-based definitions.

It is quite interesting that the relation R unifies the different proposals for the definition of quantitative information flow when they are cast as comparison problems and

[10] It can be also shown that $\neg R(M_{spec}, M_2)$ and $\neg R(M_{spec}, M_3)$, that is, M_2 and M_3 are R-incomparable with M_{spec}.

their distributions are universally quantified. As remarked in Section IV, R naturally expresses the fact that one program is more secure than the other, and it could be argued that it is the essence of quantitative information flow.

Researchers have also proposed definitions of quantitative information flow that do not fit the models studied in this paper. These include the definition based on the notion of *belief* [8], and the ones that take the maximum over the low security inputs [19], [16].[11]

Despite the staggering complexity made apparent in this paper, recent attempts have been made to (more) precisely infer quantitative information flow (without universally quantifying over the distribution as in our approach). These methods are based on the idea of *counting*. As remarked in Section III-C, quantitative information flow is closely related to counting problems, and several attempts have been made to reduce quantitative information flow problems to them.[12] For instance, Newsome et al. [26] reduce the inference problem to the #SAT problem and apply off-the-shelf #SAT solvers. To achieve scalability, they sacrifice both soundness and completeness by only computing information flow from one execution path. Backes et al. [1] also propose a counting-based approach that involves self composition. However, unlike our method, they use self composition repeatedly to find a new solution (i.e., more than a bounded number of times), and so their results do not contradict the negative results of this paper.

VI. Conclusion

We have investigated the hardness and possibilities of precisely checking and inferring quantitative information flow according to the various definitions proposed in literature. Specifically, we have considered the definitions based on the Shannon entropy, the min entropy, the guessing entropy, and channel capacity.

We have shown that comparing two programs on which has the larger flow according to these definitions is not a k-safety problem for any k, and therefore that it is not possible to reduce the problem to a safety problem via self composition. The result is in contrast to non-interference which is a 2-safety problem. We have also shown a complexity theoretic gap with non-interference by proving the #P-hardness of the comparison problems and coNP-completeness of non-interference, when restricted to loop-free boolean programs.

We have also shown a positive result that checking if the entropy-based quantitative information flow of one program is larger than that of another for all distributions is a 2-safety problem, and that it is also coNP-complete when restricted to loop-free boolean programs.

We have done this by proving a surprising result that universally quantifying the distribution in the comparison problem for the entropy-based definitions is equivalent to a simple 2-safety relation. Motivated by the result, we have proposed a novel approach to precisely checking quantitative information flow that reduces the problem to a safety problem via self composition. Our method checks the quantitative information flow of a program for all distributions against that of a specification program having the desired level of security.

[11] It is actually possible to show that the relation R refines these notions in the same sense as Theorem 4.5, but the other direction is not guaranteed to hold.

[12] Note that our results only show that, *restricted to loop-free boolean programs*, the comparison problems can be *reduced from* #SAT, and they do not show how to reduce them (or more general cases) *to* #SAT or other counting problems.

Acknowledgment

We would like to thank Takeshi Tsukada for important insights and useful discussions that motivated this work. We also thank the anonymous reviewers for their useful comments. This work was supported by MEXT KAKENHI 20700019 and 20240001, and Global COE Program "CERIES."

References

[1] M. Backes, B. Köpf, and A. Rybalchenko. Automatic discovery and quantification of information leaks. In *IEEE Symposium on Security and Privacy*, pages 141–153. IEEE Computer Society, 2009.

[2] T. Ball and S. K. Rajamani. The SLAM project: debugging system software via static analysis. In *POPL*, pages 1–3, 2002.

[3] G. Barthe, P. R. D'Argenio, and T. Rezk. Secure information flow by self-composition. In *CSFW*, pages 100–114. IEEE Computer Society, 2004.

[4] D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar. The software model checker Blast. *STTT*, 9(5-6):505–525, 2007.

[5] D. Clark, S. Hunt, and P. Malacaria. Quantified interference for a while language. *Electr. Notes Theor. Comput. Sci.*, 112:149–166, 2005.

[6] D. Clark, S. Hunt, and P. Malacaria. Quantitative information flow, relations and polymorphic types. *J. Log. Comput.*, 15(2):181–199, 2005.

[7] D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.

[8] M. R. Clarkson, A. C. Myers, and F. B. Schneider. Belief in information flow. In *CSFW*, pages 31–45. IEEE Computer Society, 2005.

[9] M. R. Clarkson and F. B. Schneider. Hyperproperties. In *CSF*, pages 51–65. IEEE Computer Society, 2008.

[10] E. S. Cohen. Information transmission in computational systems. In *SOSP*, pages 133–139, 1977.

[11] Á. Darvas, R. Hähnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In D. Hutter and M. Ullmann, editors, *SPC*, volume 3450 of *Lecture Notes in Computer Science*, pages 193–209. Springer, 2005.

[12] D. E. R. Denning. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1982.

[13] C. Flanagan and J. B. Saxe. Avoiding exponential explosion: generating compact verification conditions. In *POPL*, pages 193–205, 2001.

[14] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[15] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *POPL*, pages 58–70, 2002.

[16] B. Köpf and D. Basin. An information-theoretic model for adaptive side-channel attacks. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 286–296, New York, NY, USA, 2007. ACM.

[17] K. R. M. Leino. Efficient weakest preconditions. *Inf. Process. Lett.*, 93(6):281–288, 2005.

[18] P. Li and S. Zdancewic. Downgrading policies and relaxed noninterference. In J. Palsberg and M. Abadi, editors, *POPL*, pages 158–170. ACM, 2005.

[19] P. Malacaria. Assessing security threats of looping constructs. In M. Hofmann and M. Felleisen, editors, *POPL*, pages 225–235. ACM, 2007.

[20] P. Malacaria and H. Chen. Lagrange multipliers and maximum information leakage in different observational models. In *PLAS '08: Proceedings of the third ACM SIGPLAN workshop on Programming languages and analysis for security*, pages 135–146, New York, NY, USA, 2008. ACM.

[21] J. L. Massey. Guessing and entropy. In *ISIT '94: Proceedings of the 1994 IEEE International Symposium on Information Theory*, page 204, 1994.

[22] S. McCamant and M. D. Ernst. Quantitative information flow as network flow capacity. In R. Gupta and S. P. Amarasinghe, editors, *PLDI*, pages 193–205. ACM, 2008.

[23] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *SP '94: Proceedings of the 1994 IEEE Symposium on Security and Privacy*, page 79, Washington, DC, USA, 1994. IEEE Computer Society.

[24] K. L. McMillan. Lazy abstraction with interpolants. In T. Ball and R. B. Jones, editors, *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2006.

[25] D. A. Naumann. From coupling relations to mated invariants for checking information flow. In *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Proceedings*, pages 279–296, Hamburg, Germany, Sept. 2006.

[26] J. Newsome, S. McCamant, and D. Song. Measuring channel capacity to distinguish undue influence. In *Proceedings of the Fourth ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, Dublin, Ireland, June 2009.

[27] A. Sabelfeld and A. C. Myers. A model for delimited information release. In K. Futatsugi, F. Mizoguchi, and N. Yonezaki, editors, *ISSS*, volume 3233 of *Lecture Notes in Computer Science*, pages 174–191. Springer, 2003.

[28] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[29] G. Smith. On the foundations of quantitative information flow. In *FOSSACS '09: Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures*, pages 288–302, Berlin, Heidelberg, 2009. Springer-Verlag.

[30] T. Terauchi and A. Aiken. Secure information flow as a safety problem. In C. Hankin and I. Siveroni, editors, *SAS*, volume 3672 of *Lecture Notes in Computer Science*, pages 352–367. Springer, 2005.

[31] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

[32] H. Unno, N. Kobayashi, and A. Yonezawa. Combining type-based analysis and model checking for finding counterexamples against non-interference. In V. C. Sreedhar and S. Zdancewic, editors, *PLAS*, pages 17–26. ACM, 2006.

[33] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

APPENDIX A.
SUPPORTING DEFINITIONS AND LEMMAS

We define some abbreviations.

Definition A.1: $\mu(x) \triangleq \mu(X = x)$

We use this notation whenever the correspondences between random variables and their values are clear.

For convenience, we sometimes use large letters H, L, O, etc. to range over boolean variables as well as generic random variables.

For simplicity, we often compute the Shannon-entropy based quantitative information flow for programs that do not have low security inputs. For such programs, the equation *SE* from Definition 2.4 can be simplified as follows.

Lemma A.2:

$$\begin{aligned} SE[\mu](M) &= \mathcal{I}[\mu](O; H) \\ &= \mathcal{H}[\mu](O) \end{aligned}$$

We note the following property of deterministic programs [5].

Lemma A.3: *For M deterministic,*

$$SE[\mu](M) = \mathcal{I}[\mu](O; H|L) = \mathcal{H}[\mu](O|L)$$

The following lemma is used to show that we can generate a boolean formula that has exactly the desired number of solutions in polynomial (actually, linear) time.

Lemma A.4: *Let k be an integer such that $0 \leq k \leq 2^{|\overrightarrow{x}|} - 1$. Then, a boolean formula that has exactly k assignments over the variables $\overrightarrow{x}$ can be computed in time linear in $|\overrightarrow{x}|$.*

Proof: We define a procedure iter that returns the boolean formula. Below, $\overrightarrow{x} = x_1, x_2, \ldots$, i.e., x_i is the ith variable.

$$\begin{array}{rl} \text{iter}(\epsilon, 0) & = \mathsf{false} \\ \text{iter}(0\ell, i) & = x_i \wedge (\text{iter}(\ell, i-1)) \\ \text{iter}(1\ell, i) & = x_i \vee (\text{iter}(\ell, i-1)) \end{array}$$

Here, ϵ is an empty string. Let ℓ_k be a $|\overrightarrow{x}|$-bit binary representation of k. We prove that iter$(\ell_k, |\overrightarrow{x}|)$ returns a boolean formula that has exactly k assignments by induction on the number of variables, that is, $|\overrightarrow{x}|$.

- $|\overrightarrow{x}| = 1$
  - $k = 0$
    iter$(0, 1)$ returns $x_1 \wedge \mathsf{false}$, that is, false. false has no satisfying assignment.
  - $k = 1$
    iter$(1, 1)$ returns $x_1 \vee \mathsf{false}$, that is, x_1. x_1 has only one satisfying assignment.
- $|\overrightarrow{x}, x'|$
  - $k < 2^{|\overrightarrow{x}, x'|-1}$
    Let 0ℓ be a binary representation of k. iter$(0\ell, |\overrightarrow{x}, x'|)$ returns $x' \wedge$ iter$(\ell, |\overrightarrow{x}|)$. By induction hypothesis, iter$(\ell, |\overrightarrow{x}|)$ has k satisfying assignments for $\overrightarrow{x}$. It follows that $x' \wedge$ iter$(\ell, |\overrightarrow{x}|)$ has just k satisfying assignments, because $\mathsf{false} \wedge$ iter$(\ell, |\overrightarrow{x}|)$ has no assignment and $\mathsf{true} \wedge$ iter$(\ell, |\overrightarrow{x}|)$ has just k assignments.
  - $k \geq 2^{|\overrightarrow{x}|}$
    Let 1ℓ be a binary representation of k. iter$(1\ell, |\overrightarrow{x}, x'|)$ returns $x' \vee$ iter$(\ell, |\overrightarrow{x}|)$. ℓ is a binary representation of $k - 2^{|\overrightarrow{x}|}$. By induction hypothesis, iter$(\ell, |\overrightarrow{x}|)$ has $k - 2^{|\overrightarrow{x}|}$ satisfying assignments for $\overrightarrow{x}$. It follows that $x' \vee$ iter$(\ell, |\overrightarrow{x}|)$ has just k satisfying assignments, because $\mathsf{false} \vee$ iter$(\ell, |\overrightarrow{x}|)$ has just $k - 2^{|\overrightarrow{x}|}$ assignments and when $x' = \mathsf{true}$, $x' \vee$ iter$(\ell, |\overrightarrow{x}|)$ has just $2^{|\overrightarrow{x}|}$ assignments.

■

We frequent the following property of logarithmic arithmetic when proving statements concerning the Shannon entropy.

Lemma A.5: *Let p and q be numbers such that $p, q \in [0, 1]$. Then, we have $p \log \frac{1}{p} + q \log \frac{1}{q} \geq (p+q) \log \frac{1}{p+q}$.*

Proof: Because $\frac{p+q}{p} \geq 1$ and $\frac{p+q}{q} \geq 1$, it follows that,

$$\begin{array}{l} p \log \frac{1}{p} + q \log \frac{1}{q} - (p+q) \log \frac{1}{p+q} \\ \quad = p \log \frac{1}{p} - p \log \frac{1}{p+q} + q \log \frac{1}{q} - q \log \frac{1}{p+q} \\ \quad = p \log \frac{p+q}{p} + q \log \frac{p+q}{q} \\ \quad \geq 0 \end{array}$$

■

Appendix B.
Omitted Proofs

Theorem 2.6: *Let M be a program that takes high-security input H, low-security input L, and returns low-security output O. Then, M is non-interferent if and only if $\forall \mu. SE[\mu](M) = 0$.*

Proof: Recall that M is non-interferent iff for any $h, h' \in \mathbb{H}$ and $\ell \in \mathbb{L}$, $M(h, \ell) = M(h', \ell)$.

- ($\Rightarrow$) Suppose that M is non-interferent. Then, by Lemma A.3,

$$\begin{array}{rcl} SE[\mu](M) & = & \mathcal{I}[\mu](O; H|L) \\ & = & \mathcal{H}[\mu](O|L) \\ & = & \sum_o \sum_\ell \mu(o, \ell) \log \frac{\mu(\ell)}{\mu(o, \ell)} \\ & = & \sum_o \sum_\ell \mu(o, \ell) \log \frac{\mu(\ell)}{\mu(\ell)} \\ & = & 0 \end{array}$$

  The last step follows from the fact that non-interference implies $\mu(\ell) = \mu(o, \ell)$.
- ($\Leftarrow$) Suppose that M is interferent. Then, there must be h_0 and h_1 such that $M(h_0, \ell') = o_0$, $M(h_1, \ell') = o_1$, and $o_0 \neq o_1$. Pick a probability function μ such that $\mu(h_0, \ell') = \mu(h_1, \ell') = \frac{1}{2}$. Then, by Lemma A.3,

$$\begin{array}{rcl} SE[\mu](M) & = & \mathcal{I}[\mu](O; H|L) \\ & = & \mathcal{H}[\mu](O|L) \\ & = & \sum_o \sum_\ell \mu(o, \ell) \log \frac{\mu(\ell)}{\mu(o, \ell)} \\ & = & \mu(o_0, \ell') \log \frac{\mu(\ell')}{\mu(o_0, \ell')} \\ & & \quad + \mu(o_1, \ell') \log \frac{\mu(\ell')}{\mu(o_1, \ell')} \\ & = & \frac{1}{2} \log 2 + \frac{1}{2} \log 2 \\ & = & 1 \end{array}$$

  Therefore, there exists μ such that $SE[\mu](M) \neq 0$, and we have the conclusion.

■

We note the following equivalence of *CC* and *ME*[U] for programs without low security inputs [29].

Lemma B.1: *Let M be a program without low security input. Then, $ME[U](M) = CC(M)$.*

The min-entropy-based quantitative information flow with uniformly distributed high security input has the following property [29].

Lemma B.2: *Let M be a program without low security input and $\mathbb{O}$ be the output of M. Then, $ME[U](M) = \log(|\mathbb{O}|)$.*

Theorem 3.2: *$C_{ME}[U]$ is not a k-safety property for any $k > 0$.*

Proof: For contradiction, suppose $C_{ME}[U]$ is a k-safety property. Let M and M' be programs having same input domain such that $(M, M') \notin C_{ME}[U]$. Then, it must be the case that there exist $T \subseteq [\![M]\!]$ and $T' \subseteq [\![M']\!]$ such that $|T| \leq k$, $|T'| \leq k$, and $\forall M_c, M_c'. T \subseteq [\![M_c]\!] \wedge T' \subseteq [\![M_c']\!] \Rightarrow (M_c, M_c') \notin C_{ME}[U]$.

Let

$$T = \{(h_1, o_1), (h_2, o_2), \ldots, (h_i, o_i)\}$$
$$T' = \{(h'_1, o'_1), (h'_2, o'_2), \ldots, (h'_j, o'_j)\}$$

where $i, j \leq k$. Now, we construct new programs $\bar{M}$ and $\bar{M'}$ as follows.

$$\begin{array}{ll} \bar{M}(h_1) = o_1 & \bar{M'}(h'_1) = o'_1 \\ \bar{M}(h_2) = o_2 & \bar{M'}(h'_2) = o'_2 \\ \cdots & \cdots \\ \bar{M}(h_i) = o_i & \bar{M'}(h'_j) = o'_j \\ \bar{M}(h_{i+1}) = o & \bar{M'}(h'_{j+1}) = o'_{j+1} \\ \bar{M}(h_{i+2}) = o & \bar{M'}(h'_{j+2}) = o'_{j+2} \\ \cdots & \cdots \\ \bar{M}(h_n) = o & \bar{M'}(h'_n) = o'_n \end{array}$$

where

- o'_{j+1}, o'_{j+2}, $\ldots$, and o'_n are distinct,
- $\{o'_1, o'_2, \ldots, o'_j\} \cap \{o'_{j+1}, \ldots, o'_n\} = \emptyset$,
- $\{h_1, \ldots, h_n\} = \{h'_1, \ldots, h'_n\}$, and
- $n = 2k$.

The number of outputs of the program $\bar{M'}$ is greater than or equal to the number of the outputs of the program $\bar{M}$. Hence, by Lemma B.2, we have $(\bar{M}, \bar{M'}) \in C_{ME}[U]$. But, $T \subseteq [\![\bar{M}]\!]$ and $T' \subseteq [\![\bar{M'}]\!]$. This leads to a contradiction. ■

Definition B.3:

$$In(\mu, X, x) = |\{x' \in X \mid \mu(x') \geq \mu(x)\}|$$

Intuitively, $In(\mu, X, x)$ is the order of x defined in terms of μ.

Lemma B.4:

$$\begin{array}{ll} \mathcal{G}[\mu](X) & = \Sigma_{1 \leq i \leq |X|} i\mu(x_i) \\ & = \Sigma_{x \in X} In(\mu, X, x)\mu(x) \end{array}$$

Proof: Trivial. ■

Lemma B.5: *Let μ be a function such that $\mu : \mathbb{D} \to [0, 1]$. Let P and Q be sets such that $P \cup Q = \mathbb{D}$ and $P \cap Q = \emptyset$. Then, we have $\sum_{x \in \mathbb{D}} In(\mu, \mathbb{D}, x)\mu(x) \geq \sum_{p \in P} In(\mu, P, p)\mu(p) + \sum_{q \in Q} In(\mu, Q, q)\mu(q)$.*

Proof: Trivial. ■

Definition B.6: *Let M be a function such that $M : \mathbb{A} \to \mathbb{B}$. For any $o \in B$, we define $M^{-1}(o)$ to mean*

$$M^{-1}(o) = \{i \in \mathbb{A} \mid o = M(i)\}$$

Theorem 3.3: *$C_{GE}[U]$ is not a k-safety property for any $k > 0$*

Proof: For contradiction, suppose $C_{GE}[U]$ is a k-safety property. Let M and M' be programs having the same input domain such that $(M, M') \notin C_{GE}[U]$. Then, it must be the case that there exist $T \subseteq [\![M]\!]$ and $T' \subseteq [\![M']\!]$ such that $|T| \leq k$, $|T'| \leq k$, and $\forall M_c, M_c'. T \subseteq [\![M_c]\!] \wedge T' \subseteq [\![M_c']\!] \Rightarrow (M_c, M_c') \notin C_{GE}[U]$.

Let

$$T = \{(h_1, o_1), (h_2, o_2), \ldots, (h_i, o_i)\}$$
$$T' = \{(h'_1, o'_1), (h'_2, o'_2), \ldots, (h'_j, o'_j)\}$$

where $i, j \leq k$. Now, we construct new programs $\bar{M}$ and $\bar{M'}$ as follows.

$$\begin{array}{ll} \bar{M}(h_1) = o_1 & \bar{M'}(h'_1) = o'_1 \\ \bar{M}(h_2) = o_2 & \bar{M'}(h'_2) = o'_2 \\ \cdots & \cdots \\ \bar{M}(h_i) = o_i & \bar{M'}(h'_j) = o'_j \\ \bar{M}(h_{i+1}) = o & \bar{M'}(h'_{j+1}) = o'_{j+1} \\ \bar{M}(h_{i+2}) = o & \bar{M'}(h'_{j+2}) = o'_{j+2} \\ \cdots & \cdots \\ \bar{M}(h_{i+j}) = o & \bar{M'}(h'_{j+i}) = o'_{j+i} \\ \bar{M}(h_{i+j+1}) = o_r & \bar{M'}(h'_{j+i+1}) = o'_r \\ \cdots & \cdots \\ \bar{M}(h_n) = o_r & \bar{M'}(h'_n) = o'_r \end{array}$$

where

- $o \neq o_r$,
- $\{o_1, o_2, \ldots, o_i\} \cap \{o, o_r\} = \emptyset$,
- o'_{j+1}, o'_{j+2}, $\ldots$, o'_{j+i}, and o'_r are distinct,
- $\{o'_1, o'_2, \ldots, o'_j\} \cap \{o'_{j+1}, \ldots, o'_{j+i}, o'_r\} = \emptyset$,
- $\{h_1, \ldots, h_n\} = \{h'_1, \ldots, h'_n\}$, and
- $n = 2k$.

We compare the guessing-entropy-based quantitative information flow of the two programs.

$$\begin{array}{ll} & GE[U](\bar{M'}) - GE[U](\bar{M}) \\ = & \frac{|\mathbb{H}|}{2} - \frac{1}{2|\mathbb{H}|} \sum_{o' \in M'(\mathbb{H})} |M'^{-1}(o')|^2 \\ & - \frac{|\mathbb{H}|}{2} + \frac{1}{2|\mathbb{H}|} \sum_{o \in M(\mathbb{H})} |M^{-1}(o)|^2 \\ = & \frac{1}{2|\mathbb{H}|} \sum_{o \in M(\mathbb{H})} |M^{-1}(o)|^2 \\ & - \frac{1}{2|\mathbb{H}|} \sum_{o' \in M'(\mathbb{H})} |M'^{-1}(o')|^2 \\ = & \frac{1}{2|\mathbb{H}|} (\sum_{o_x \in \{o_1, \ldots, o_i\}} |M^{-1}(o_x)|^2 \\ & \quad + |M^{-1}(o)|^2 + |M^{-1}(o_r)|^2) \\ & - \frac{1}{2|\mathbb{H}|} (\sum_{o'_x \in \{o'_1, \ldots, o'_j\}} |M'^{-1}(o'_x)|^2 \\ & \quad + \sum_{o'_y \in \{o'_{j+1}, \ldots, o'_{j+i}\}} |M'^{-1}(o'_y)|^2 \\ & \quad + |M'^{-1}(o'_r)|^2) \end{array}$$

By lemma B.5, we have

$$\sum_{o_x \in \{o_1, \ldots, o_i\}} |M^{-1}(o_x)|^2 \leq \sum_{o'_y \in \{o'_{j+1}, \ldots, o'_{j+i}\}} |M'^{-1}(o'_x)|^2$$

and

$$|M^{-1}(o)|^2 \leq \sum_{o'_x \in \{o'_1, \ldots, o'_j\}} |M'^{-1}(o'_x)|^2$$

Trivially, we have

$$|M'^{-1}(o'_r)|^2 = |M^{-1}(o_r)|^2$$

$T(\phi) =$
  if ϕ
    then $O_f := \mathsf{true}; \overrightarrow{O} := \overrightarrow{H}$
    else $O_f := \mathsf{false}; \overrightarrow{O} := \overrightarrow{\mathsf{false}}$

where O_f and $\overrightarrow{O}$ are distinct.

Figure 3. Boolean formula encoding by boolean program

As a result, we have

$$GE[U](\bar{M}') - GE[U](\bar{M}) \geq 0$$

Recall that $\bar{M}$ and $\bar{M}'$ have the same counterexamples T and T', that is, $T \subseteq [\![\bar{M}]\!]$ and $T' \subseteq [\![\bar{M}']\!]$. However, we have $(\bar{M}, \bar{M}') \in C_{GE}[U]$. This leads to a contradiction. ■

Theorem 3.4: *C_{CC} is not a k-safety property for any $k > 0$.*

Proof: Straightforward from Lemma B.1 and Theorem 3.2. ■

Lemma B.7: *Let $\overrightarrow{H}$ be distinct boolean variables, ϕ be a boolean formula over $\overrightarrow{H}$, and n be the number of satisfying assignments for ϕ. If n is less than $2^{|\overrightarrow{H}|}$, then the number of the outputs of the boolean program $T(\phi)$ defined in Figure 3 is equal to $n + 1$.*

Proof: Trivial. ■

Lemma B.8: *Let $\overrightarrow{H}$ be distinct variables and ϕ be a boolean formula over $\overrightarrow{H}$. Then, the number of assignments for ϕ can be computed by executing an oracle that decides whether programs are in $C_{ME}[U]$ at most $3 * (|\overrightarrow{H}| + 1) + 2$ times.*

Proof: First, we define a procedure that returns the number of solutions for ϕ.

Let $B(j) = \psi \wedge H'$ where ψ is a formula over $\overrightarrow{H}$ having j assignments and H' is a boolean variable such that $H' \notin \{\overrightarrow{H}\}$. Note that by Lemma A.4, such ψ can be generated in linear time.

Then, we invoke the following procedure where T is defined in Figure 3.

$\ell = 0;$
$r = 2^{|\overrightarrow{H}|};$
$n = (\ell + r)/2;$
while $\neg((T(\phi \wedge H'), T(B(n))) \in C_{ME[U]}$
    and $(T(B(n)), T(\phi \wedge H')) \in C_{ME}[U])$
  if $(T(\phi \wedge H'), T(B(n))) \in C_{ME}[U]$
    then $\{\ell = n; n = (\ell + r)/2;\}$
    else $\{r = n; n = (\ell + r)/2;\}$
return n

Note that when the procedure terminates, we have $ME[U](T(B(n))) = ME[U](T(\phi \wedge H'))$, and so by Lemma B.2 and Lemma B.7, n is the number of satisfying assignments to ϕ.

We show that the procedure iterates at most $|\overrightarrow{H}|+1$ times. To see this, note that every iteration in the procedure narrows the range between r and ℓ by one half. Because $r - \ell$ is bounded by $2^{|\overrightarrow{H}|}$, it follows that the procedure iterates at most $|\overrightarrow{H}| + 1$ times. Hence, the oracle $C_{ME}[U]$ is accessed $3 * (|\overrightarrow{H}| + 1) + 2$ times, and this proves the lemma. ■

Theorem 3.6: *#P $\subseteq$ FP$^{C_{ME}[U]}$*

Proof: Straightforward by Lemma B.8 and the fact that #SAT, the problem of counting the number of solutions to a boolean formula, is #P-complete. ■

Lemma B.9: *Let $\overrightarrow{H}$ and H' be distinct variables and ϕ and ϕ' be boolean formulas over $\overrightarrow{H}$. Let $M \equiv O := \phi \wedge H'$ and $M' \equiv O := \phi' \wedge H'$. Then, we have $\#SAT(\phi) \leq \#SAT(\phi')$ iff $GE[U](M) \leq GE[U](M')$.*

Proof: By the definition,

$$\begin{aligned} GE[U](M) &= \mathcal{G}(H) - \mathcal{G}(H|O) \\ &= \tfrac{1}{2}(|\overrightarrow{H}|) + \tfrac{1}{2} - \textstyle\sum_o \sum_{1 \leq i \leq |\overrightarrow{H}|} iU(h_i, o) \\ &= \tfrac{|\overrightarrow{H}|}{2} \\ &\quad - \tfrac{1}{2|\overrightarrow{H}|}(|M^{-1}(\mathsf{true})|^2 + |M^{-1}(\mathsf{false})|^2) \end{aligned}$$

Therefore,

$$GE[U](M) \leq GE[U](M')$$

iff

$$\begin{aligned} &|M^{-1}(\mathsf{true})|^2 + |M^{-1}(\mathsf{false})|^2 \\ &\quad \geq |M'^{-1}(\mathsf{true})|^2 + |M'^{-1}(\mathsf{false})|^2 \end{aligned}$$

But, trivially, the latter holds iff

$$\#SAT(\phi) \leq \#SAT(\phi')$$

■

Lemma B.10: *Let $\overrightarrow{H}$ and H' be distinct variables and ϕ be a boolean formula over $\overrightarrow{H}$. Then, the number of assignments for ϕ can be computed by executing an oracle that decides whether programs are in $C_{GE}[U]$ at most $3 * (|\overrightarrow{H}| + 1) + 2$ times.*

Proof: First, we define a procedure that returns the number of solutions for ϕ.

Let $B(j) = \psi \wedge H'$ where ψ is a formula over $\overrightarrow{H}$ having j assignments and H' is a boolean variable such that $H' \notin \{\overrightarrow{H}\}$. Note that by Lemma A.4, such ψ can be generated in linear time.

$\ell = 0;$
$r = 2^{|\overrightarrow{H}|};$
$n = (\ell + r)/2;$
while $\neg(O := \phi \wedge H', O := B(n)) \in C_{GE[U]}$
    and $(O := B(n), O := \phi \wedge H') \in C_{GE}[U])$
  if $(O := \phi \wedge H', O := B(n)) \in C_{GE}[U]$
    then $\{\ell = n; n = (\ell + r)/2;\}$
    else $\{r = n; n = (\ell + r)/2;\}$
return n

Note that when this procedure terminates, we have $GE[U](O := B(n)) = GE[U](O := \phi \wedge H')$, and so by Lemma B.9, n is the number of satisfying assignments to ϕ.

We show that the procedure iterates at most $|\overrightarrow{H}|+1$ times. To see this, every iteration in the procedure narrows the range between r and ℓ by one half. Because $r-\ell$ is bounded by $2^{|\overrightarrow{H}|}$, it follows that the procedure iterates at most $|\overrightarrow{H}|+1$ times. Hence, the oracle $C_{GE}[U]$ is accessed $3*(|\overrightarrow{H}|+1)+2$ times, and this proves the lemma. ■

Theorem 3.7: *#P* $\subseteq$ *FP*$^{C_{GE}[U]}$

Proof: Straightforward by Lemma B.10 and the fact that #SAT, the problem of counting the number of solutions to a boolean formula, is #P-complete. ■

Theorem 3.8: *#P* $\subseteq$ *FP*$^{C_{CC}}$

Proof: Straightforward from Lemma B.1 and Theorem 3.6. ■

Theorem 3.9: *Checking non-interference is coNP-complete for loop-free boolean programs.*

Proof: We write NI for the decision problem of checking non-interference of loop-free boolean programs. We prove by reducing NI to and from UNSAT, which is coNP-complete.

- NI $\subseteq$ UNSAT
  We reduce via self composition [3], [11]. Let M be a boolean program that we want to know if it is non-interferent. First, we make a copy of M, with each variable x in M replaced by a fresh (primed) variable x'. Call this copy M'. Let $\phi = wp(M; M', \overrightarrow{O} = \overrightarrow{O}')$, where $\overrightarrow{O} = \overrightarrow{O}'$ is the boolean formula encoding the conjunction of equalities $O_1 = O_1'$, $O_2 = O_2'$, …, $O_n = O_n'$, where $O_1, \ldots, O_n$ are the low security output variables of M. Note that ϕ can be obtained in time polynomial in the size of M. Here, instead of the rules in Figure 2, we use the optimized weakest precondition generation technique [13], [17] that generates a formula quadratic in the size of $M; M'$. Then, M is non-interferent if and only if ϕ is valid, that is, if and only if $\neg\phi$ is unsatisfiable.
- UNSAT $\subseteq$ NI
  Let ϕ be a formula that we want to know if it is unsatisfiable. We prove that the following programs is non-interferent iff ϕ is unsatisfiable. Here, all variables that appear in ϕ are high security input variables and H is a high security input variable that is distinct from variables appearing in ϕ, and O is the low security output variable.

  $$\mathsf{if}\ \phi \wedge H\ \mathsf{then}\ O := \mathsf{true}\ \mathsf{else}\ O := \mathsf{false}$$

  Trivially, if ϕ is unsatisfiable, then this program returns only false, that is, this program is non-interferent. If this program is non-interferent, then this program returns only true for any input, or returns only false for any input. However, this program can not return only true, because if $H = \mathsf{false}$ then $\phi \wedge H = \mathsf{false}$. Therefore, this program only returns false, when this program is non-interferent. That means ϕ is unsatisfiable when the program is non-interferent.

■

Definition B.11: *Let M be a function such that $M : \mathbb{A} \to \mathbb{B}$. Then, we define the image of M on $\mathbb{X} \subseteq \mathbb{A}$, $M[\mathbb{X}]$, as follows.*

$$M[\mathbb{X}] = \{o \mid o = M(x) \wedge x \in \mathbb{X}\}$$

Lemma B.12: *Let $\mathbb{H}$ be a set, and M and M' be functions whose domains contain $\mathbb{H}$. Suppose that we have $M'(h_0, l) = M'(h_1, l) \Rightarrow M(h_0, l) = M(h_1, l)$, for all h_0, h_1 in $\mathbb{H}$. Then, for all $h' \in \mathbb{H}$, we have $\{h \mid M'(h, l) = M'(h', l)\} \subseteq \{h \mid M(h, l) = M(h', l)\}$.*

Proof: Trivial. ■

Lemma B.13: *Let H, O, O', and L be distinct random variables. Let M and M' be programs. We have $(M, M') \in R$ iff for any distribution μ, $\mathcal{H}_\infty[\mu](H|O', L) \leq \mathcal{H}_\infty[\mu](H|O, L)$ where $O' = M'(H, L)$ and $O = M(H, L)$.*

Proof:

- ($\Rightarrow$)
  Suppose $R(M, M')$. We have

  $$\begin{aligned}&\mathcal{H}_\infty[\mu](H|O', L) \leq \mathcal{H}_\infty[\mu](H|O, L)\\ &\quad \text{iff } \mathcal{V}[\mu](H|O, L) \leq \mathcal{V}[\mu](H|O', L)\end{aligned}$$

  by the definition of min entropy, and

  $$\begin{aligned}&\mathcal{V}[\mu](H|O, L)\\ &= \textstyle\sum_{o\in\mathbb{O},\ell\in\mathbb{L}} \mu(o, \ell) \max_{h\in\mathbb{H}} \mu(h|o, \ell)\\ &= \textstyle\sum_{o\in\mathbb{O},\ell\in\mathbb{L}} \mu(o, \ell) \max_{h\in\mathbb{H}} \frac{\mu(h,o,\ell)}{\mu(o,\ell)}\\ &= \textstyle\sum_{o\in\mathbb{O},\ell\in\mathbb{L}} \max_{h\in\mathbb{H}} \mu(o, \ell) \frac{\mu(h,o,\ell)}{\mu(o,\ell)}\\ &= \textstyle\sum_{o\in\mathbb{O},\ell\in\mathbb{L}} \max_{h\in\mathbb{H}} \mu(h, o, \ell)\\ &= \textstyle\sum_{o\in\mathbb{O},\ell\in\mathbb{L}} \max_{h\in\{h'|o=M(h',\ell)\}} \mu(h, \ell)\end{aligned}$$

  where $\mathbb{O} = M[\{(h, \ell) \in \mathbb{H} \times \mathbb{L} \mid \mu(h, l) > 0\}]$, and $\mathbb{L}$ and $\mathbb{H}$ are sample spaces of low-security input and high-security input, respectively. Therefore, it suffices to show that

  $$\begin{aligned}&\mathcal{V}[\mu](H|O', L) - \mathcal{V}[\mu](H|O, L)\\ &= \textstyle\sum_{o'\in\mathbb{O}',\ell\in\mathbb{L}} \max_{h\in\{h'|o'=M'(h',\ell)\}} \mu(h, \ell)\\ &\quad - \textstyle\sum_{o\in\mathbb{O},\ell\in\mathbb{L}} \max_{h\in\{h'|o=M(h',\ell)\}} \mu(h, \ell)\\ &\geq 0\end{aligned}$$

  where $\mathbb{O}' = M'[\{(h, \ell) \in \mathbb{H} \times \mathbb{L} \mid \mu(h, \ell) > 0\}]$.
  For any $o \in \mathbb{O}$ and $\ell \in \mathbb{L}$, there exists h_m such that $\mu(h_m, \ell) = \max_{h\in\{h'|o=M(h',\ell)\}} \mu(h, \ell)$. Because $R(M, M')$, by Lemma B.12, we have

  $$\begin{aligned}&\{h \mid M'(h, \ell) = M'(h_m, \ell)\}\\ &\quad \subseteq \{h \mid M(h, \ell) = M(h_m, \ell)\}\end{aligned}$$

Therefore,

$$\mu(h_m, \ell) = \max_{h \in \{h' | o' = M'(h', \ell)\}} \mu(h, \ell)$$

for some $o' \in \mathbb{O}'$. Hence, each summand in $\sum_{o \in \mathbb{O}, \ell \in \mathbb{L}} \max_{h \in \{h' | o = M(h', \ell)\}} \mu(h, \ell)$ also appears in $\sum_{o' \in \mathbb{O}', \ell \in \mathbb{L}} \max_{h \in \{h' | o' = M'(h', \ell)\}} \mu(h, \ell)$. And, we have the above proposition.

- ($\Leftarrow$)
We prove the contraposition. Suppose $(M, M') \notin R$. Then, there exist h_0, h_1, ℓ, o_0, o_1 such that $M'(h_0, \ell) = M'(h_1, \ell)$, $o_0 = M(h_0, \ell)$, $o_1 = M(h_1, \ell)$, and $o_0 \neq o_1$. Pick a probability distribution μ such that $\mu(h_0, \ell) = \mu(h_1, \ell) = \frac{1}{2}$. Then, we have

$$\begin{aligned} &\mathcal{V}[\mu](H|O', L) \\ &= \textstyle\sum_{o' \in \mathbb{O}', \ell \in \mathbb{L}} \max_{h \in \{h' | o' = M(h', \ell)\}} \mu(h, \ell) \\ &= \tfrac{1}{2} \end{aligned}$$

and

$$\begin{aligned} &\mathcal{V}[\mu](H|O, L) \\ &= \textstyle\sum_{o \in \mathbb{O}, \ell \in \mathbb{L}} \max_{h \in \{h' | o = M(h', \ell)\}} \mu(h, \ell) \\ &= \tfrac{1}{2} + \tfrac{1}{2} \\ &= 1 \end{aligned}$$

Therefore, $\mathcal{H}_\infty[\mu](H|O', L) \not\leq \mathcal{H}_\infty[\mu](H|O, L)$.

■

Theorem 4.3: $R = \{(M_1, M_2) \mid \forall \mu. C_{ME}[\mu](M_1, M_2)\}$

Proof: Straightforward from Lemma B.13 and the fact that $\mathcal{H}_\infty[\mu](H|L) - \mathcal{H}_\infty[\mu](H|O, L) \leq \mathcal{H}_\infty[\mu](H|L) - \mathcal{H}_\infty[\mu](H|O', L)$ iff $\mathcal{H}_\infty[\mu](H|O, L) \geq \mathcal{H}_\infty[\mu](H|O', L)$.

■

Theorem 4.4: $R = \{(M_1, M_2) \mid \forall \mu. C_{GE}[\mu](M_1, M_2)\}$

Proof:

- $\subseteq$
Suppose $(M, M') \in R$. By the definition,

$$\begin{aligned} &GE[\mu](M) = \\ &\textstyle\sum_{\ell \in \mathbb{L}, h \in \mathbb{H}} In(\lambda h'.\mu(h', \ell), \mathbb{H}, h)\mu(h, \ell) \\ &- \textstyle\sum_{o \in \mathbb{O}, \ell \in \mathbb{L}, h \in \mathbb{H}} In(\lambda h'.\mu(h', o, \ell), \mathbb{H}, h)\mu(h, o, \ell) \end{aligned}$$

and

$$\begin{aligned} &GE[\mu](M') = \\ &\textstyle\sum_{\ell \in \mathbb{L}, h \in \mathbb{H}} In(\lambda h'.\mu(h', \ell), \mathbb{H}, h)\mu(h, \ell) \\ &- \textstyle\sum_{o' \in \mathbb{O}', \ell \in \mathbb{L}, h \in \mathbb{H}} In(\lambda h'.\mu(h', o', \ell), \mathbb{H}, h)\mu(h, o', \ell) \end{aligned}$$

where $\mathbb{O} = M[\{(h, \ell) \in \mathbb{H} \times \mathbb{L} \mid \mu(h, \ell) > 0\}]$ and $\mathbb{O}' = M'[\{(h, \ell) \in \mathbb{H} \times \mathbb{L} \mid \mu(h, \ell) > 0\}]$.
It suffices to show that

$$\begin{aligned} &\textstyle\sum_{o' \in \mathbb{O}', \ell \in \mathbb{L}, h \in \mathbb{H}} In(\lambda h'.\mu(h', o', \ell), \mathbb{H}, h)\mu(h, o', \ell) \\ &\quad \leq \textstyle\sum_{o \in \mathbb{O}, \ell \in \mathbb{L}, h \in \mathbb{H}} In(\lambda h'.\mu(h', o, \ell), \mathbb{H}, h)\mu(h, o, \ell) \end{aligned}$$

Let $o \in \mathbb{O}$ and $\ell \in \mathbb{L}$. Let $o = M(h_0, \ell) = \cdots = M(h_x, \ell)$, and let $o'_0 = M'(h_0, \ell), \ldots, o'_x = M'(h_x, \ell)$. Because $R(M, M')$, for any h' such that $M'(h', \ell) \in \{o'_0, \ldots, o'_x\}$, we have $h' \in \{h_0, \ldots, h_x\}$. Then, by Lemma B.5, we have

$$\begin{aligned} &\textstyle\sum_{h \in \mathbb{H}_\mathbb{O}} In(\lambda h'.\mu(h', o', \ell), \mathbb{H}, h)\mu(h, o, \ell) \\ &\quad \geq \textstyle\sum_{o' \in {\mathbb{O}'}_o, h \in \mathbb{H}_o} In(\lambda h'.\mu(h', o', \ell), \mathbb{H}, h)\mu(h, o', \ell) \end{aligned}$$

where

$$\begin{aligned} {\mathbb{O}'}_o &= \{o'_0, \ldots, o'_x\} \\ \mathbb{H}_o &= \{h_0, h_1, \ldots, h_x\} \end{aligned}$$

Now we prove each $\mathbb{O}_o$ constructed above are disjoint. That is, for o_1 and o_2 outputs of M such that $o_1 \neq o_2$, $\mathbb{O}_{o_1} \cap \mathbb{O}_{o_2} = \emptyset$. For a contradiction, suppose $o' \in \mathbb{O}_{o_1} \cap \mathbb{O}_{o_2}$. Then, there exist h_1 and h_2 such that $o_1 = M(h_1, \ell)$, $o' = M'(h_1, \ell)$, $o_2 = M(h_2, \ell)$, and $o' = M'(h_2, \ell)$. Since $R(M, M')$, we have $o_1 = o_2$, and it leads to a contradiction. Hence, we have for any $\ell \in \mathbb{L}$,

$$\begin{aligned} &\textstyle\sum_{o' \in \mathbb{O}', h \in \mathbb{H}} In(\lambda h'.\mu(h', o', \ell), \mathbb{H}, h)\mu(h, o', \ell) \\ &\quad \leq \textstyle\sum_{o \in \mathbb{O}, h \in \mathbb{H}} In(\lambda h'.\mu(h', o, \ell), \mathbb{H}, h)\mu(h, o, \ell) \end{aligned}$$

Therefore, it follows that

$$\begin{aligned} &\textstyle\sum_{o' \in \mathbb{O}', \ell \in \mathbb{L}, h \in \mathbb{H}} In(\lambda h'.\mu(h', o', \ell), \mathbb{H}, h)\mu(h, o', \ell) \\ &\quad \leq \textstyle\sum_{o \in \mathbb{O}, \ell \in \mathbb{L}, h \in \mathbb{H}} In(\lambda h'.\mu(h', o, \ell), \mathbb{H}, h)\mu(h, o, \ell) \end{aligned}$$

- $\supseteq$
We prove the contraposition. Suppose $(M, M') \notin R$. Then, there exist h, h', ℓ, o, o' such that
  - $M(h, \ell) = o$, $M(h', \ell) = o'$, and $o \neq o'$
  - $M'(h, \ell) = M'(h', \ell)$

Then, we can pick μ such that $\mu(h, \ell) = \mu(h', \ell) = 0.5$. We have

$$GE[\mu](M) = 1.5 - 1 = 0.5$$

and

$$GE[\mu](M') = 1.5 - 1.5 = 0$$

Therefore, we have $(M, M') \notin C_{GE}[\mu]$.

■

Theorem 4.5: $R \subseteq C_{CC}$

Proof: Let M and M' be programs such that $(M, M') \in R$. We prove $(M, M') \in C_{CC}$.
By Theorem 4.2, we have

$$\forall \mu. SE[\mu](M) \leq SE[\mu](M')$$

Now, there exists μ' such that

$$CC(M) = SE[\mu'](M)$$

Therefore,

$$SE[\mu'](M) \leq SE[\mu'](M')$$

Trivially,

$$SE[\mu'](M') \leq CC(M')$$

Therefore, we have the conclusion. ■

Theorem 4.6: *Let M_2 be a non-interferent program. Then, $R(M_1, M_2)$ iff M_1 is also non-interferent and M_1 has the same input domain as M_2.*

Proof: Straightforward from Theorems 2.6 and 4.2. ■

Theorem 4.8: *Restricted to loop-free boolean programs, R is coNP-complete.*

Proof:

- $R \subseteq$ coNP
  We prove by reducing R to UNSAT, which is coNP-complete. We reduce via self composition [3], [11]. Let M and M' be boolean programs that we want to know if they are in R. First, we make copies of M and M', with all variables in M and M' replaced by fresh (primed) variables. Call these copies M_c and M'_c. Let $\phi = wp(M; M_c; M'; M'_c, \overrightarrow{O'} = \overrightarrow{O'_c} \Rightarrow \overrightarrow{O} = \overrightarrow{O_c})$ where $\overrightarrow{O}$,$\overrightarrow{O_c}$,$\overrightarrow{O'}$, and $\overrightarrow{O'_c}$ are the low security outputs of M,M_c,M', and M'_c, respectively. Note that ϕ can be obtained in time polynomial in the size of M and M'. Here, like in Theorem 3.9, we use the optimized weakest precondition generation technique [13], [17] to generate a formula quadratic in the size of $M; M_c; M'; M'_c$. Then, $(M, M') \in R$ if and only if ϕ is valid, that is, if and only if $\neg\phi$ is unsatisfiable.
- coNP $\subseteq R$
  We prove by reducing NI to R, because NI is coNP-complete by Theorem 3.9. We can check the non-interference of M by solving $R(M, M')$ where M' is non-interferent and have the same input domain as M by Theorem 4.6. Note that such M' can be constructed in polynomial time. Therefore, we have coNP $\subseteq R$.

■